高等学校计算机应用规划教材

ASP.NET 4.5
网站开发实例教程

- Web基础知识
- ASP.NET开发环境及项目文件结构
- 配置管理与页面管理
- 内置对象
- ASP.NET服务器控件
- 页面切换与导航
- 样式、主题与母版页
- ADO.NET数据访问技术
- ASP.NET数据绑定技术
- ASP.NET AJAX、jQuery技术
- Web Services技术
- 在ASP.NET中使用XML



耿　超　编著

清华大学出版社

高等学校计算机应用规划教材

ASP.NET 4.5
网站开发实例教程

耿　超　编著

清华大学出版社

北　　京

内 容 简 介

本书全面讲述了 ASP.NET 网站建设的基本原理和技术。全书共分为 14 章，深入介绍了使用 Visual Studio 2012 集成环境建立 ASP.NET 网站的相关技术，包括 Web 基础知识，ASP.NET 开发环境的下载、安装、组成和使用，ASP.NET 网页框架语言，ASP.NET 程序结构、页面管理，ASP.NET 的常用内置对象，ASP.NET 配置管理，C#语法基础，ASP.NET 服务器控件，页面切换与导航，样式、主题与母版页，ADO.NET 数据访问技术，ASP.NET 中的数据绑定，ASP.NET AJAX，jQuery，Web 服务，在 ASP.NET 中使用 XML 等内容。书中所有知识都结合具体实例进行介绍，涉及的程序代码都给出了详细的注释，可以使读者轻松领会使用 ASP.NET 进行 Web 程序开发的精髓，从而快速提高开发技能。最后，介绍了一个网上书店综合实例，巩固前面所学。

本教程内容丰富、结构合理、思路清晰、语言简练流畅、示例翔实。它主要面向网站开发初学者，适合作为各种网站建设培训班的培训教材、高等院校相关专业的教材，还可作为 ASP.NET 应用开发人员的参考书。

本书的电子教案、习题答案和实例源文件可以到 http://www.tupwk.com.cn 网站下载。

图书在版编目(CIP)数据

ASP.NET 4.5 网站开发实例教程 / 耿超 编著. —北京：清华大学出版社，2015(2018.6 重印)
(高等学校计算机应用规划教材)
ISBN 978-7-302-40229-9

Ⅰ. ①A… Ⅱ. ①耿… Ⅲ. ①网页制作工具—程序设计—高等学校—教材 Ⅳ. ①TP393.092

中国版本图书馆 CIP 数据核字(2015)第 101270 号

责任编辑：胡辰浩　袁建华
装帧设计：孔祥峰
责任校对：成凤进
责任印制：刘海龙

出版发行：清华大学出版社
网　址：http://www.tup.com.cn，http://www.wqbook.com
地　址：北京清华大学学研大厦 A 座　　邮　编：100084
社 总 机：010-62770175　　邮　购：010-62786544
投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015, zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn，010-62794504

印 刷 者：北京富博印刷有限公司
装 订 者：北京市密云县京文制本装订厂
经　销：全国新华书店
开　本：185mm×260mm　　印　张：24　　字　数：554 千字
版　次：2015 年 6 月第 1 版　　印　次：2018 年 6 月第 3 次印刷
印　数：5051~7000
定　价：56.00 元

产品编号：056516-02

前　　言

随着微软对.NET 平台的大力支持，ASP.NET 成为目前使用最为普遍的 Web 项目开发技术之一。随着移动互联技术的井喷式发展，微软推出了最新的 Visual Studio 2012 来支持更加多样化的开发。

ASP.NET 使用 VS 平台进行 Web 应用程序的开发，具有先天性的优势。开发人员能够快速使用 ASP.NET 提供的控件和技术进行网站开发，同时还支持移动应用程序开发。

为了让想要系统学习 ASP.NET 开发技术的人员可以在短时间内掌握这门技术，特编写了本书。本书以实用为原则，选择了当前使用最为广泛的 Visual Studio 2012 作为开发平台，全面、系统、深入地介绍了 ASP.NET 开发的各项技术。书中提供了大量的实例、小型开发模块和综合项目案例供读者实战演练。各章后面还提供了习题，帮助读者巩固本章所学。相信学习了本书内容后，读者能够较好地掌握 ASP.NET 开发技术，并具备基本的项目开发能力。

本书具有如下特色：

1. 技术全面，涵盖广泛

本书几乎涉及 ASP.NET 应用开发的所有常见技术，全面涵盖了 ASP.NET 基础、窗体控件、数据库开发、Web 页面开发 Ajax 技术、jQuery、主题、母版、CSS 样式等内容。

2. 技术新颖，紧跟趋势

本书紧跟 Web 开发的趋势，书中介绍的很多内容都是当前在.NET 开发中应用非常广泛，而且非常新颖和流行的技术。掌握了本书内容，读者可以快速地胜任实际的开发工作。

3. 由浅入深，循序渐进

本书从基础开始讲解，然后层层推进，逐步深入到 ASP.NET 开发的各种高级技术，最后介绍了一个网上书店的综合项目案例。全书学习梯度平滑，适合各个层次的读者阅读。

4. 实例丰富，代码详解

本书在讲解知识点时都给出了大量的示例或实例供读者理解和实战演练，最后一章介绍了一个网上书店的综合项目案例开发过程。这些案例都提供了完整的代码，并且给代码进行了详细的注释和剖析。

5. 规范开发，提高效率

本书详细介绍了进行规范的应用程序开发的要求，如设计需求分析文档的编写、类图的编写等，便于读者编写出更加规范和易于阅读的程序，从而提高开发效率。

本书共分 14 章，内容涵盖了 ASP.NET 的各个方面，主要内容如下。

第 1 章，ASP.NET 4.5 概述。主要介绍了静态网站与动态网站、HTTP 协议、Web 服务器和浏览器、C/S 结构与 B/S 结构、Web 工作原理、ASP.NET 发展历史、ASP.NET 的开发环境、定制 IDE、创建和运行 ASP.NET 应用程序等内容。

第 2 章，ASP.NET 网页框架语言。主要介绍了 Web 基本技术、XHTML 的基本格式、XHTML 常用标记及标记属性等内容。

第 3 章，ASP.NET Web 技术简介。主要介绍了 ASP.NET 文件类型、ASP.NET 文件夹、ASP.NET 页面代码模式、页面的往返与处理机制、页面的生命周期及事件、ASP.NET 页面指令、ASP.NET 的常用内置对象、ASP.NET 配置管理等内容。

第 4 章，C#语法基础。主要介绍了 C#语言特点、如何建立 C#程序、类、对象、属性成员与方法成员、参数传递、C#数据类型、运算符、流程控制语句、继承、接口、托管、命名空间等内容。

第 5 章，ASP.NET 服务器控件。ASP.NET 服务器控件概述、HTML 元素和 HTML 服务器控件、标准服务器控件、验证控件、用户控件等内容。

第 6 章，页面切换与导航。主要介绍了通过超链接方式、控件的 PostBackUrl 属性方式、Response.Redirect 或 Server.Transfer 方式进行页面切换、页面间的数据传递，还介绍了 ASP.NET 网站导航功能、利用站点地图和 SiteMapPath 控件实现网站导航、通过 Menu 控件实现导航、通过 TreeView 控件实现导航等内容。

第 7 章，样式、主题与母版页。主要介绍了 CSS 语言基本概念、页面布局、主题、母版页等内容。

第 8 章，ADO.NET 数据访问技术。主要介绍了 ADO.NET 基本概念，使用 Connection 连接数据库的方法，使用 Command、DataAdapter 对象执行数据库命令对数据记录进行增删改查操作等内容。

第 9 章，ASP.NET 中的数据绑定。主要介绍了数据绑定中的基本概念、单值和列表控件的数据绑定、通过 GridView、DataList 和 FormView 控件进行数据绑定等内容。

第 10 章，ASP.NET AJAX 技术。主要介绍了 ASP.NET AJAX 基本概念、ScriptManager 控件、UpdatePanel 控件、UpdateProgress 控件、Timer 控件、ASP.NET AJAX Control Extenders 扩展控件等内容。

第 11 章，jQuery 技术。主要介绍了 jQuery 基本概念和语法基础、通过 jQuery 修改 DOM、使用 jQuery 的效果方法、jQuery 和有效性验证及实用提示等内容。

第 12 章，Web 服务技术。主要介绍了 Web Services 的概念、JSON 和支持 Ajax 的 Web 服务、创建和使用支持 Ajax 的 Web 服务、支持 Ajax 的 Web 服务的应用程序体系结构等内容。

第 13 章，在 ASP.NET 中使用 XML。主要介绍了 XML 的基本概念、使用 ADO.NET 访问 XML、使用.NET 的 XML 类访问 XML、XML 的应用和发展前景等内容。

第 14 章，介绍了一个网上书店的开发。

本教程内容丰富、结构合理、思路清晰、语言简练流畅、示例翔实。每一章的引言部分概述了本章的学习目标。在每一章的正文中，结合所讲述的关键技术和难点，穿插了大量极富实用价值的示例。每一章末尾都安排了有针对性的思考和练习，思考题有助于读者巩固所学的基本概念，练习题有助于培养读者的实际动手能力、增强对基本概念的理解和实际应用能力。

本书主要面向网站建设初学者，适合作为各种网站建设培训班的培训教材、高等院校相关专业的教材及 ASP.NET 应用开发人员的参考书。

本书的电子教案、习题答案和实例源文件可以到 http://www.tupwk.com.cn 网站下载。

除封面署名的作者外，参加本书编写的人员还有黄果、李琼琼、周倩芸、韩高洁、郭纳、李泽峰、张洪军、王显波、陈华东、陈明卫、林帅、王德学、苗润田、覃仕益、龙光华、陈发辉、吴彪、刘雄、李常等人。由于作者水平有限，本书难免有不足之处，欢迎广大读者批评指正。我们的信箱是 huchenhao@263.net，电话是 010-62796045。

作　者

2015 年 3 月

目　　录

第1章　ASP.NET 4.5概述

本章介绍了网站建设的基本原理、流程和网站开发工具——ASP.NET 技术的基本概况。作为主流之一的 Web 开发技术，ASP.NET 基于 Microsoft 公司的.NET 框架，支持 C # 和 VB.NET 语言，是目前主要的网站开发平台之一。通过本章的学习，读者能够了解有关 Web 的基础知识，ASP.NET 技术的发展及其优缺点，ASP.NET 开发环境的组成，创建和运行 ASP.NET 应用程序等知识。

本章的学习目标：

- 掌握 Web 基础概念，如静态网站、动态网站、HTTP 协议、Web 服务器和浏览器、C/S 和 B/S 结构、Web 工作原理等。
- 了解 ASP.NET 的发展历史、优点及与 ASP 技术的区别。
- 掌握 ASP.NET 开发环境的组成。
- 了解开发动态网站的一般流程，能够创建简单的动态网站。

1.1　Web 基础知识

互联网的快速发展给人们的工作、学习和生活带来了重大变化，人们可以利用网络处理数据、获取信息，提高工作效率。在互联网开发所涉及的众多技术中，最为关键的技术之一就是网站建设技术。通过本节的学习，读者将了解到网站制作过程中常见的基础知识。

1.1.1　静态网站与动态网站

1. 静态网站

传统的网站一般是采用静态网页技术制作的静态网站。在静态网站中所有的内容以 HTML 语言编写，存储在静态网页文件中，文件扩展名为：.htm、.html、.shtml 和.xml 等。网页上可以有 GIF 动画、Flash 动画和滚动字幕等，但这些效果只是视觉上的，并未构成动态网页。这里所讨论的静态网站中的“静”是指网页内容在用户发出请求之前就已经生成了(这就是用户每次总能看到相同页面的原因)，Web 服务器只负责保存和传递 HTML 文件，而不进行额外处理，用户只能浏览网站所提供的信息，如图 1-1 所示。

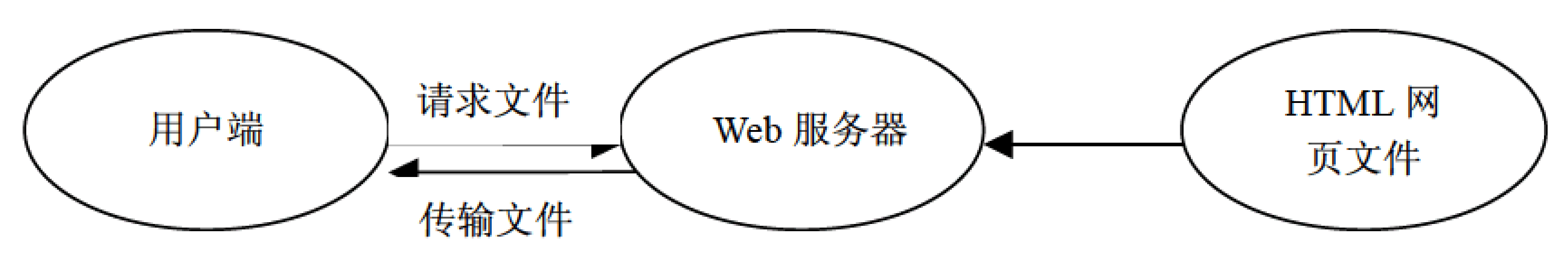


图 1-1　静态网站模型

静态网站中的网页，内容相对稳定，不需要通过数据库工作。对于 Web 服务器来说，处理负担不大。因此，静态网站具有容易被搜索引擎检索、访问速度比较快的优点。

静态网站的致命弱点是不易维护，为了不断更新网页内容，网站管理者必须不断地重复制作 HTML 页面。随着网站内容和信息量的日益增长，维护工作将变得十分艰巨。因此，静态网站往往适用于数据不多、网页比较固定、更新不频繁的情况。更新较少的展示型网站一般采用静态网站技术搭建。

2. 动态网站

在静态网站中用户可以阅读、发布大量信息，但如果用户想拥有自己的聊天室——享受与朋友在一起自由讨论的乐趣，建立网上商店——实现自己的创业梦想等等，就必须采用动态网站技术进行交互式 Web 体验。

什么是动态网站呢？所谓“动”，并不是指网页上的 GIF 动画图片，而是指用户与网站的互动性。

动态网站的概念现在还没有统一标准，但一般应满足以下几点特征。

(1) 交互性

动态网站中的网页会根据用户的要求和选择而改变和响应。网站管理员只需要掌握计算机基本操作方法，就可以方便、及时地更新网站内容；浏览网站的用户可以在网站中进行查询、留言等操作。可见，动态网站技术大大增加了管理员与网站、客户与网站的互动性。

(2) 通过数据库进行架构

在动态网站中，网络管理员除了要设计网页视觉效果，还要设计数据库和程序代码来使网站具有更好的交互功能。例如，购物网站中含有大量的商品种类和数量信息，为了方便查找，就应当基于数据库在网页上实现自动搜索。现在广泛使用的网上交易系统、在线采购系统、商务交流系统等都是基于数据库来提供的。

(3) 在服务器端运行，方便更新

在服务器端运行的程序，会随不同客户、不同要求返回不同的页面，网站管理员再也不用像静态网站那样手动更新网页文档，可以大大提高工作效率，如图 1-2 所示。

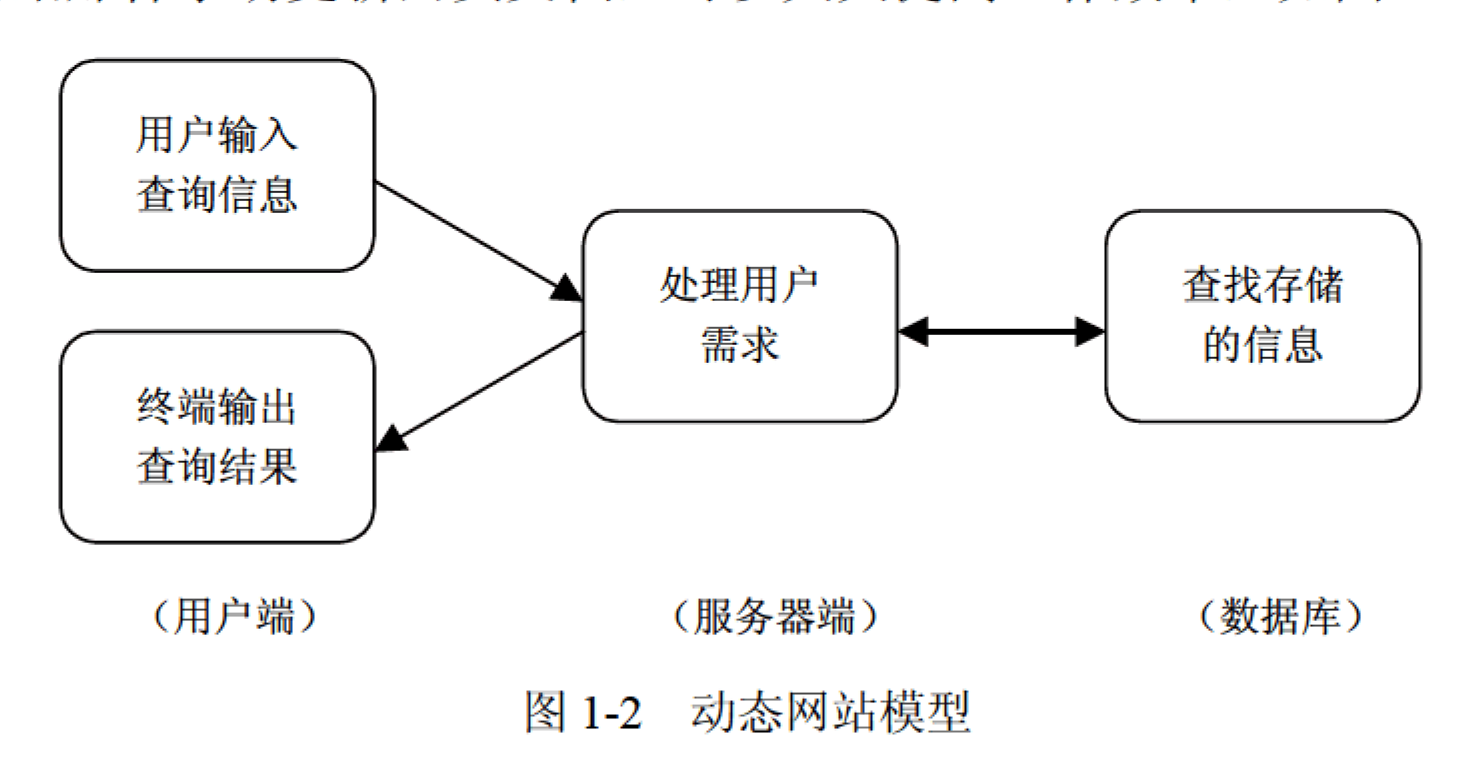


图 1-2　动态网站模型

由上述特征可以看出，静态网站和动态网站的主要区别在于：静态网站内容是在用户

发出请求之前就预先生成的，而动态网站内容则是在用户发出请求之后服务器根据用户提供的指令执行产生的。

动态网站是目前主流的网站技术，动态网站在发出请求之后才生成内容，这有两个明显的优点。

首先，服务器端可以根据用户提交的请求以及所提供的参数值生成需要的页面内容。例如，在一个电子商务网站提交用户名和密码，那么用户将看到的下一个页面就是动态生成的页面，它包含了该用户的账户信息，简而言之，就像登录淘宝网之后可以看到个人账户一样。

其次，服务器端可以根据最新的可用信息设置它所生成的页面内容。例如，很多网站都有正在使用这个网站的用户数量的显示。用户数量的值是实时信息，是在 Web 服务器接收用户请求时获取的。

可以说，动态网站的 Web 服务器不仅负责传递网页文件，它更是一个完成信息处理的执行平台。例如，一个购物网站，由两部分组成，一个是用户部分(也叫前台客户端)，一个是管理员部分(也叫后台服务器端)。前台客户端提供的功能包括：新用户注册，已注册用户的登录，用户对商品的查询、浏览，用户对商品的在线购买等。当一个新用户在前台注册成功，后台服务器就必须在数据库的用户表中添加相应的用户信息；当已注册用户想查看自己购物车中的物品，后台服务器就必须从数据库中找出该用户已挑选出物品的详细信息，产生输出结果，反馈到用户浏览器中。

静态网站和动态网站各有特点，搭建网站采用动态还是静态技术主要取决于网站的功能需求和内容的多少。如果网站功能比较简单，内容更新量不是很大，采用静态网站的方式会更简单。反之，一般要采用动态网站技术来实现。

静态网站可以使用 Frontpage 或 Dreamweaver 等网页编辑工具来建立，而动态网站需要使用服务器端网页技术，如本书介绍的 ASP.NET 来搭建。

1.1.2　HTTP 协议

HTTP 协议(HyperText Transfer Protocol，即：超文本传输协议)是用于在 WWW 服务器和浏览器之间传输超文本的传输协议。HTTP 是一个应用层协议，由请求和响应构成，是一个标准的客户端/服务器(C/S)模型，如图 1-3 所示。

HTTP 是一个无状态的协议。它可以使浏览器显示更加高效，使网络传输量减少。它不仅保证了计算机正确快速地传输超文本文档，还确定传输文档中的哪一部分，以及哪部分内容首先显示(如文本先于图形)等。

HTTP 协议的特点可概括如下。

- 支持客户/服务器模式：支持基本认证和安全认证。
- 简单快速：客户向服务器请求服务时，只需传送请求方法和路径。常用的请求方法有 GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于 HTTP 协议简单，使得 HTTP 服务器的程序规模小，因而通信速度很快。

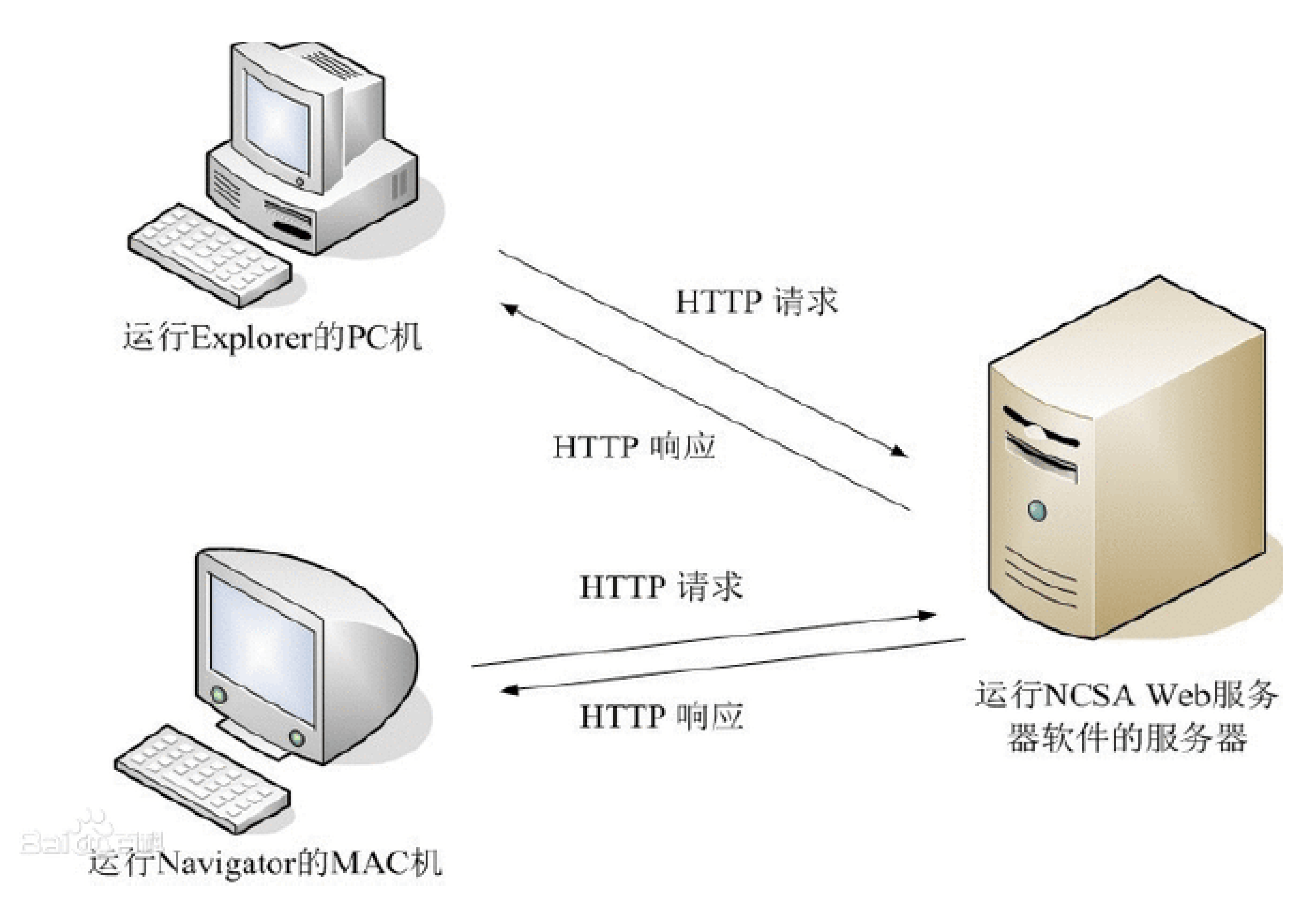


图 1-3　HTTP 协议请求应答模型

- 灵活：HTTP 允许传输任意类型的数据对象。正在传输的类型由 Content-Type 加以标记。
- HTTP 0.9 和 HTTP 1.0 使用非持续连接：限制每次连接只处理一个请求，服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。
- HTTP 1.1 使用持续连接：不必为每个 Web 对象创建一个新的连接，一个连接可以传送多个对象。
- 无状态：HTTP 协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

1.1.3　Web 服务器和浏览器

Internet 上的服务器也称为 Web 服务器，是一台在 Internet 上具有独立 IP 地址的计算机，可以向 Internet 上的客户机提供网上信息浏览服务，如 WWW、Email 和 FTP 等各种 Internet 服务。当 Web 浏览器(客户端)连到服务器上并请求文件时，服务器将处理该请求并将文件反馈到该浏览器上，附带的信息会告诉浏览器如何查看该文件(即文件类型)。服务器使用 HTTP(超文本传输协议)与客户机浏览器进行信息交流，这就是常把它们称为 HTTP 服务器的原因。

WWW 是 Internet 的多媒体信息查询工具，是 Internet 上近年才发展起来的服务，也是发展最快和目前用的最广泛的服务。Web 服务器不仅能够存储信息，还能在用户通过 Web 浏览器提供的信息的基础上运行脚本和程序。正是因为有了 WWW 工具，才使得近年来 Internet 迅速发展，且用户数量飞速增长。

总的来说，用户通过浏览器向 Web 服务器提出访问请求，而 Web 服务器可以向发出请求的浏览器提供文档。这个访问模型如图 1-4 所示。

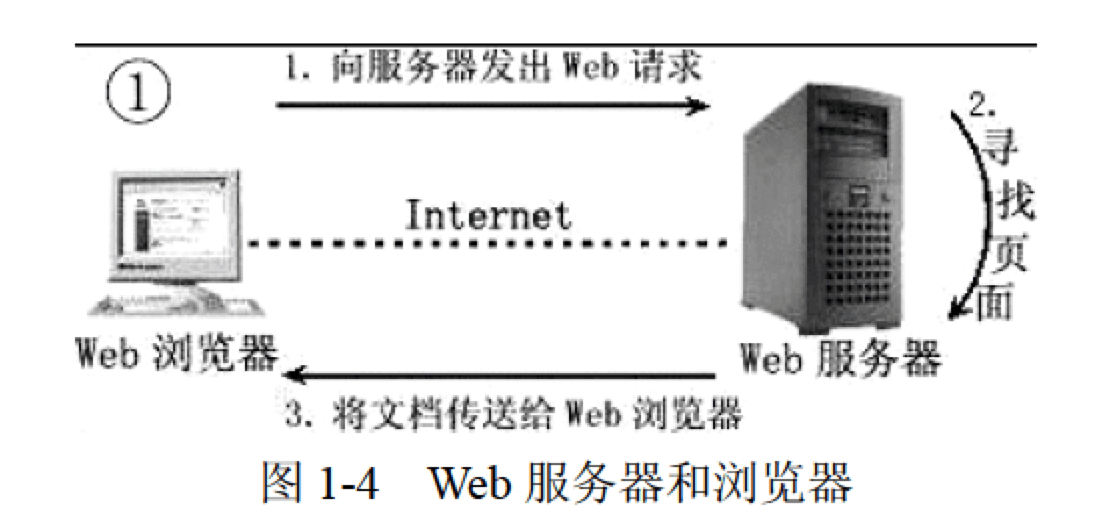


图 1-4　Web 服务器和浏览器

1.1.4　C/S 结构与 B/S 结构

1. C/S 结构

C/S 结构，即大家熟知的客户机和服务器结构，如图 1-5 所示。它是软件系统体系结构，通过它可以充分利用两端硬件环境的优势，将任务合理分配到 Client 端和 Server 端，降低了系统的通信开销。大多数应用软件系统都是 Client/Server 形式的两层结构，由于现在的软件应用系统正在向分布式的 Web 应用发展，Web 和 Client/Server 应用都可以进行同样的业务处理，应用不同的模块共享逻辑组件。因此，内部的和外部的用户都可以访问新的和现有的应用系统，通过现有应用系统中的逻辑可以扩展出新的应用系统。

Client 和 Server 常常分别处在相距很远的两台计算机上，Client 程序的任务是将用户的要求提交给 Server 程序，再将 Server 程序返回的结果以特定的形式显示给用户；Server 程序的任务是接收客户程序提出的服务请求，进行相应的处理，再将结果返回给客户程序。

传统的 C/S 体系结构虽然采用的是开放模式，但这只是系统开发一级的开放性，在特定的应用中无论是 Client 端还是 Server 端都还需要特定的软件支持。由于没能提供用户真正期望的开放环境，C/S 结构的软件需要针对不同的操作系统开发不同版本的软件，加之产品的更新换代十分快，已经很难适应百台电脑以上局域网用户同时使用。而且代价高，效率低。

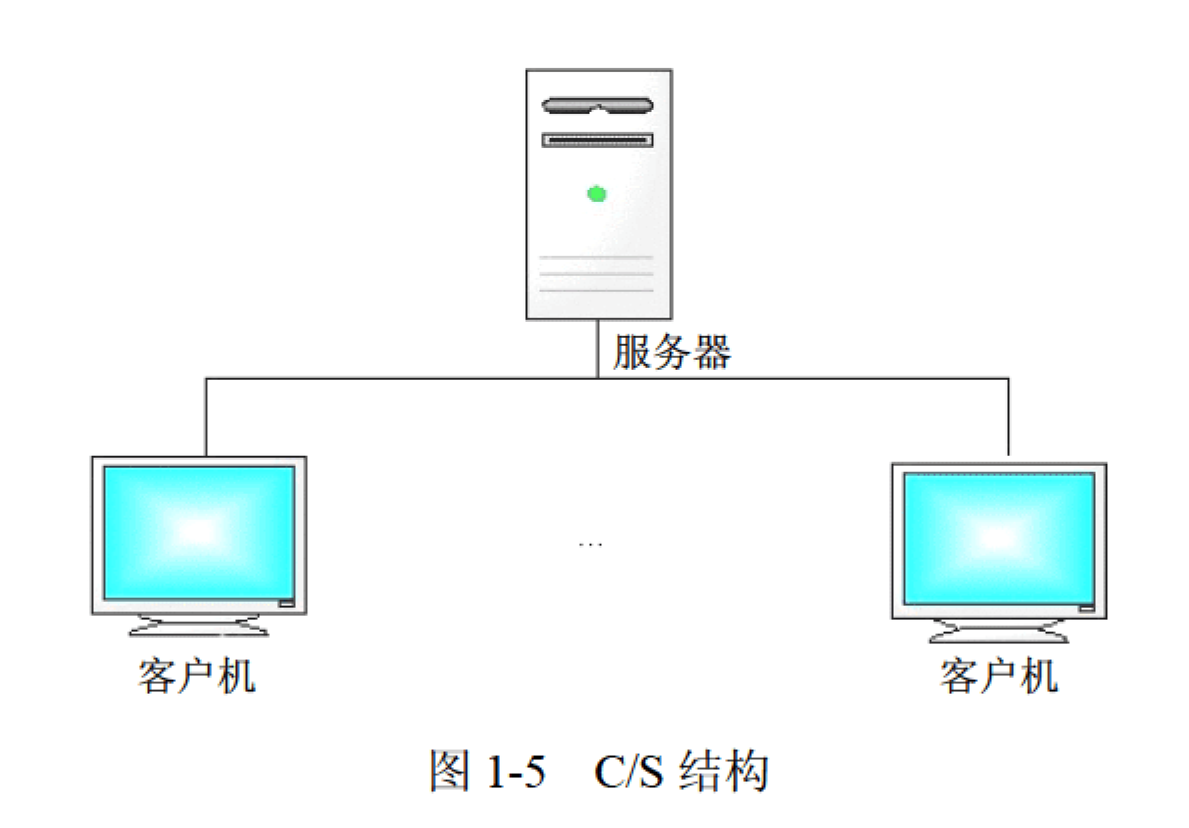


图 1-5　C/S 结构

2. B/S 结构

B/S 结构(Browser/Server，即：浏览器/服务器模式)，是 Web 兴起后的一种网络结构模式，Web 浏览器是客户端最主要的应用软件。这种模式使客户端得到统一，将系统功能实现的核心部分集中到服务器上，简化了系统的开发、维护和使用。客户机上只须安装一个浏览器(Browser)，如 Netscape Navigator 或 Internet Explorer，服务器只须安装 SQL Server、Oracle、MYSQL 等数据库。浏览器通过 Web Server 同数据库进行数据交互。

B/S 最大的优点就是可以在任何地方进行操作而不用安装任何专门的软件，只要有一台能上网的计算机就能使用，客户端零安装、零维护。系统的扩展非常容易。

B/S 结构的使用越来越多，特别是由需求推动了 Ajax 技术的发展，它的程序也能在客户端电脑上进行部分处理，从而大大地减轻了服务器的负担；并增加了交互性，能进行局部实时刷新。

- 维护和升级方式简单。当前，软件系统的改进和升级越发频繁，B/S 架构的产品明显体现着更为方便的特性。对一个稍微大一点单位来说，系统管理人员如果需要在几百甚至上千部电脑之间来回奔跑，效率和工作量是可想而知的，但 B/S 架构的软件只须管理服务器就行了，所有的客户端只是浏览器，根本无须做任何的维护。无论用户的规模有多大，有多少分支机构都不会增加任何维护升级的工作量，所有的操作只须针对服务器进行；如果是异地，只须把服务器连接专网即可，实现远程维护、升级和共享。所以客户机越来越“瘦”，而服务器越来越“胖”，这是将来信息化发展的主流方向。今后，软件升级和维护会越来越容易，而使用起来会越来越简单，这对人力、物力、时间、费用的节省是显而易见的。因此，维护和升级革命的方式是“瘦”客户机，“胖”服务器。
- 成本降低，选择更多。大家都知道 Windows 在桌面电脑上几乎一统天下，浏览器成为了标准配置，但在服务器操作系统上 Windows 并不是处于绝对的统治地位。当前的趋势是凡使用 B/S 架构的应用管理软件，只须安装在 Linux 服务器上即可，而且安全性高。所以服务器操作系统的选择是很多的，不管选用哪种操作系统都可以让大部分人使用 Windows 作为桌面电脑操作系统不受影响，这就使得最流行免费的 Linux 操作系统快速发展起来。Linux 除了操作系统是免费的以外，连数据库也是免费的，这种选择非常盛行。

  例如，很多人每天上“新浪”网，只要安装了浏览器就可以了，并不需要了解“新浪”的服务器用的是什么操作系统，而事实上大部分网站确实没有使用 Windows 操作系统，但用户的电脑本身安装的大部分是 Windows 操作系统。
- 应用服务器运行数据负荷较重。由于 B/S 架构管理软件只安装在服务器端(Server)上，网络管理人员只须管理服务器就行了，用户界面主要事务逻辑在服务器(Server)端完全通过 WWW 浏览器实现，极少部分事务逻辑在前端(Browser)实现，所有的客户端只有浏览器，网络管理人员只须做硬件维护。但是，应用服务器运行数据负荷较重，一旦发生服务器“崩溃”等问题，后果不堪设想。因此，许多单位都备有数据库存储服务器，以防万一。

1.1.5　Web 工作原理

WWW 的工作基于客户机/服务器计算模型，由 Web 浏览器(客户机)和 Web 服务器(服务器)构成，两者之间采用超文本传送协议(HTTP)进行通信，HTTP 协议的作用原理包括 4 个步骤：连接、请求、应答、关闭。

Web 的工作原理是：用户单击超链接或在浏览器中输入地址后，此时浏览器将该信息转换成标准的 HTTP 请求发送给 Web 服务器。当 Web 服务器接收到 HTTP 请求后，根据请求内容查找所需信息资源。找到相应资源后，Web 服务器将该部分资源通过标准的 HTTP

响应发送回浏览器。最后浏览器接收到响应后将 HTML 文档显示出来。工作原理如图 1-6 所示。

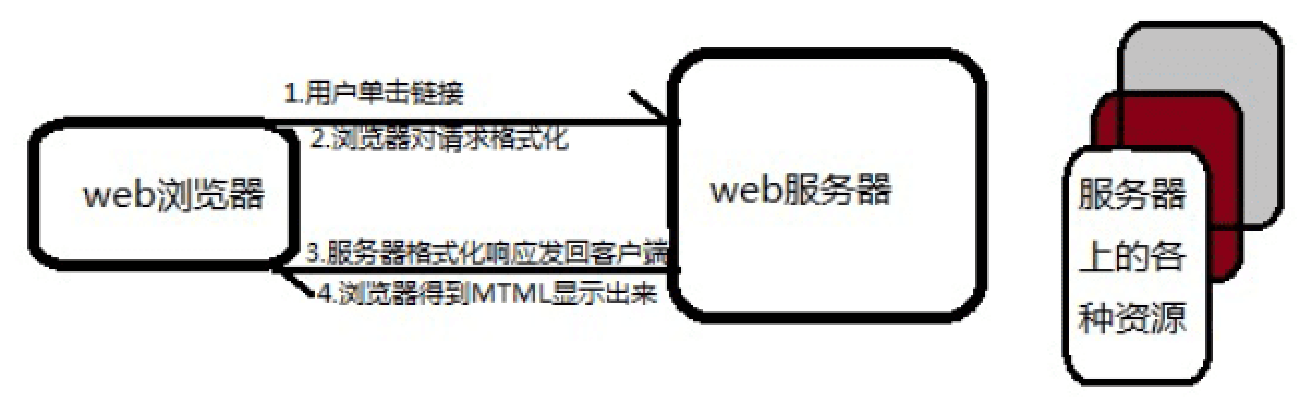


图 1-6　Web 工作原理图

1.2　ASP.NET 简介

1.2.1　ASP.NET 的历史

自从.NET Framework 1.0 在 2002 年初首次发布以来，Microsoft 花了大量精力和时间来开发 ASP.NET，它是.NET Framework 的一部分，可以用来构建复杂 Web 应用程序。首次发布意味着从过去的 Microsoft 技术向构建 ASP (Active Server Page，即：活动服务器页面，现在人们常称之为传统 ASP)Web 站点的飞跃。与传统 ASP 相比，ASP.NET 1.0 及相关的 Visual Studio .NET 的引入给开发人员带来了如下好处。

- 页面显示与代码清楚地分开。使用传统 ASP 时，编程逻辑常常散布在整个页面的 HTML 中，使得后面对页面的修改比较困难。
- 开发模型更接近于桌面应用程序的编程方式。这样很多 Visual Basic 桌面程序员可以轻松地转换到 Web 应用程序。
- 它有一个功能丰富的开发工具(称为 Visual Studio .NET)，开发人员可以通过它可视化地创建和编写 Web 应用程序代码。
- 有几种面向对象的编程语言可供选择，其中 Visual Basic .NET 和 C#(读作 C-Sharp)是目前最流行的两种语言。
- 它可以访问整个.NET Framework，这意味着 Web 开发人员首次拥有了一种统一且容易的方式，来使用数据库、文件、E-mail、网络工具等许多高级功能。

尽管 ASP.NET 远优于旧模型，但使用它也意味着构建应用程序时的复杂性以及所需知识量的增加，所以它对于许多新的程序员来说，更难上手。

在 2002 年首次发布以后，Microsoft 在 2003 年发布了.NET Framework 的另一个版本(称为.NET 1.1)和开发 IDE(称为 Visual Studio .NET 2003)。尽管在架构和开发工具方面都有了不少新的改进，但很多人还是习惯把这些看做是初始版本的一个服务包。

2005 年 11 月，Microsoft 发布了 Visual Studio 2005 和 ASP.NET 2.0。让全球许多开发人员感到惊喜的是，Microsoft 又大大改进和扩展了产品，增加了许多功能和工具来帮助降低 ASP.NET 1.0 所带来的复杂性。新的向导和智能控件减少了构建应用程序所需的代码，

降低了新开发人员的学习难度，并且提高了开发效率。

2007 年 11 月，Microsoft 发布了 Visual Studio 2008 和 ASP.NET 3.5 框架，之后又在 2010 年 3 月发布了 Visual Studio 2010 和 ASP.NET 4，这两个版本都增加了许多新功能，包括 LINQ(参见第 14 章)，Ajax 框架整合(参见第 10 章)，ADO.NET Entity Framework(参见第 14 章)，和 jQuery(参见第 11 章)。

目前的版本是 Visual Studio 2012 和 ASP.NET 4.5，它是在已成功发行的 Visual Studio 2010 和 ASP.NET 4 基础之上构建的，它保留了很多令人喜爱的功能，并增加了一些其他领域的新功能和工具。

在本书接下来的章节中，将会介绍如何使用 Visual Studio Express 2012 for Web(它是 Microsoft 为 ASP.NET Web 应用程序设计的免费开发工具，也是 Visual Studio 2012 完整套装的一部分)构建功能完全的 ASP.NET Web 站点。本书将引导读者了解构建功能完全的、数据库驱动的 Web 站点的过程，从第 2 章的基本 Web 站点开始，直到第 14 章将它部署到生产环境中。

本书的示例站点和所有实例都是用 Visual Studio Express 2012 for Web (VSEW)构建的，因此需要把它安装到开发机器上。第 1.3 节将介绍 VSEW 开发环境的组成。安装并运行 VSEW 后，就学习如何创建第一个 Web 站点，接着介绍 VSEW 的许多功能。

1.2.2　ASP 与 ASP.NET 的区别

用户使用 ASP 做开发时，常面临的问题是：如果应用程序要有强大的功能，就需要编写大量的代码，所有的功能都必须用代码来实现。虽然使用一些小技巧可以实现代码的共享，但是仍然存在大量嵌入 HTML 的代码，难于管理。

从更深的层次来研究 ASP 和 ASP.NET，可以发现它们之间的区别主要在于以下几个方面。

1. 效率

ASP 是一个脚本编程环境，只能用 VBScript 或者 JavaScript 这样的非模块化语言来编写。当 ASP 程序完成之后，在每次请求的时候解释执行。这就意味着它在使用其他语言编写的大量组件的时候会遇到困难，并且无法实现对操作系统的底层操作。由于它是解释执行的，所以速度和效率要远远低于编译执行的程序。而 ASP.NET 则是建立在.NET Framework 之上的，它可以使用 Visual Basic、C++这样的模块化程序设计语言，并且它在第一次执行时进行了编译，之后的执行无须重新编译就可以直接运行，所以速度和效率比 ASP 会提高很多。

2. 可重用性

在写 ASP 应用程序时，ASP 代码和 HTML 混合在一起。只要有需要，就可以在任意的一个位置插入一段代码来实现特定的功能。这种方法表面上看起来很方便，但在实际的工作中会产生大量繁琐的页面，很难读懂，导致代码维护很困难。虽然可以使用 include

指令尽量让程序模块化，但是仍然不是一个最终彻底的解决方案。而 ASP.NET 则可以实现代码和内容的完全分离，使得前面提到的问题迎刃而解。

3. 代码量

ASP 需要对所有的要实现的功能通过编写代码来实现。例如，所有的 ASP 程序员都遇到过这样的情况：为了保证一个用户数据提交页面的友好性，当用户输入错误的时候会显示错误的位置，并尽量把用户原来的输入在控件中显示出来。这样的一个应用需要程序员编写大量的代码来实现。虽然它实在是一个非常简单的功能，但在其他的应用程序中根本不需要编写这么多的代码就可以实现。在 ASP.NET 中，程序员会发现只要预先说明，这样的功能 ASP.NET 就可以自动实现。所以相对来说，要实现同样的功能，使用 ASP.NET 比使用 ASP 的代码量要小得多。

1.2.3　ASP.NET 的优点

通过上面对 ASP 缺点的介绍，可以很容易地理解 ASP.NET 已经解决了这些问题。事实的确如此，并且 ASP.NET 所提供的不仅仅是这些。要了解 ASP.NET 的所有特性，下面先来看看设计 ASP.NET 的 4 个主要目标：

- 使代码更清晰。
- 提高可部署性、可伸缩性、安全性以及可靠性。
- 为不同的浏览器和设备提供更好的支持。
- 支持一种全新的 Web 应用程序。

使用 ASP.NET 来进行开发，主要有以下优点。

1. 多语言

ASP 曾在脚本引擎方面受到了限制，特别是 VBScript 和 JScript。而.NET Framework 本质上就支持多语言，因此用户可以使用任何需要的语言。在默认情况下，Visual Basic .NET、C#和 JScript .NET 都附带了 CLR(它们都是经过编译的)，另外还可以使用许多第三方语言，如 Perl、COBOL 等。此外，Visual Studio .NET 增加了对 Visual C++的支持和 Java 实现(称为 J# .NET)。由于多语言支持是.NET Framework 的一部分，因此无论使用何种语言都没有关系。很明显，从用户的观点来看，保持某种程度上的兼容性可能是最好不过的，但对于架构而言，语言的使用是没有任何限制的。

多语言支持的作用并不仅仅限于可使用什么语言，同时还在于如何使用这些语言。通过它可以用某种语言编写组件，而用另一种语言来使用(或重用)这些组件。例如，用 C#编写基于服务器的控件，然后在 Visual Basic .NET 中继承这些控件，接着在 JScript .NET (或任何.NET 支持的语言)中继续继承这些控件。

2. 服务器处理

如果用户做过 Visual Basic 编程工作，就会发现要掌握新的 ASP.NET 服务器控件是相当容易的，但如果以前仅使用过 ASP，就会在学习 ASP.NET 服务器控件的初始阶段出现

一些混淆错误。不过即使这样也不用担心，因为它们是非常容易理解和使用的——它们只是和 ASP 有很大的不同。

使用 ASP 的一大问题在于页面简单地定义了一个大函数，该函数开始于页面的开头，结束于页面的末尾。不管页面的内容是纯 HTML 还是 ASP 生成的 HTML，它都是按照页面顺序显示的。因此，逻辑代码依赖于它在页面中的位置，并且无法将 HTML 控件作为目标，除非将它们作为流的一部分进行显示。开发人员所做的任何工作都需要编写代码，包括 HTML 元素的输出。

而 ASP.NET 就解决了这个问题，方法是为控件引入一个声明的基于服务器的模型。这对 ASP 开发人员来说是一个截然不同的概念，因为控件在服务器上声明，可以在服务器上编程，也可以是客户端驱动的事件。为把一个普通的 HTML 控件变成一个服务器控件，只需将 runat="server"作为一个特性添加到代码中。例如：

```
<input id="FirstName" type="text" runat="server" >
```

这是一个标准的 HTML 控件，增加了 runat 特性后，就可以利用服务器端的代码对该控件进行编程了。例如，如果要将该控件置于一个窗体中，并且将该窗体提交回同一个页面中，就可以在服务器端代码中做以下修改。

```
Dim PersonFirstName As String
PersonFirstName = FirstName.Text
```

控件在服务器上运行，将允许程序员使用 ID 特性来直接识别它。这样代码就具有更高的可读性，原因在于不必引用窗体的内容或者将这些内容复制到变量中。同时直接引用控件也就更加自然了，从而使页面的开发更加简单。如果用户做过 Visual Basic 或 VBA 编程工作，对这些概念就不会太陌生。

但是如果用户以前只用 ASP 编写过脚本，理解上述内容就比较吃力，不过这仅仅是因为 ASP.NET 用不同的方式处理往返于浏览器的页面内容。如果用户了解数据库访问技术，就应该使用过对象、调用方法以及设置过属性，而这些与 ASP.NET 服务器控件并无多大差别。

3. Web Form 控件

将现有的 HTML 控件转换为服务器端的控件是比较容易的，不过这种方法仍然存在以下几个问题。

- 一致性：开发者仍然不得不接受一些 HTML 控件的非常不直观的固有特征。例如，有没有一个 INPUT 标记符用于单行文本项？有没有一个 TEXTAREA 标识符用于多行文本项？指定行和列的单个控件真的更有意义吗？
- 用户经验：如何轻松地编写站点，向诸如 IE 这样的浏览器传送丰富的内容，同时保留对低级别浏览器的兼容性？HTML 是无法根据浏览器改变其内容的，而要达到这个目的，必须编写代码。

- 设备：如何编写站点来应付除浏览器之外的其他设备呢？如今像 WAP 手机、PDA 甚至电冰箱这样的设备都有浏览器。与浏览器问题一样，都必须为其手工编写代码。

为了解决这些问题，Microsoft 公司创建了一组服务器控件，它们都用 asp:前缀标识。ASP.NET 服务器控件通过以下方式来处理上述问题。

- 提供一致的命名标准。例如，所有文本输入栏都由 TextBox 控件处理。对于不同的模式(如多行文本、密码等)，只须指定相应的特性即可。
- 提供一致的属性。所有的服务器控件使用一组一致的属性，这样更便于记忆。例如，TextBox 控件中的 Text 栏比 Value 栏更加直观。
- 提供一致的事件模型。传统的 ASP 页面通常有大量用于处理数据传送的代码，特别是当一个页面提供多行命令时，就更是如此。而利用 ASP.NET，开发者可以为事件过程封装控件，从而给服务器端代码提供更多的结构。
- 发送纯 HTML 或者 HTML 与客户端 JavaScript。通过一个小异常，服务器控件在默认状态下会发送 HTML 3.2，提供很高的跨浏览器兼容性。对其进行修改，使之在默认状态下指向像 IE 这样的高级浏览器，这样控件就会发送 HTML 4.0 和 DHTML，提供更丰富的界面。此时用户只能看到 HTML 内容，而看不到服务器控件。
- 发送设备指定的代码。当一个浏览器发出请求时，某个控件会发送 HTML，而当 WAP 手机发出请求时，该控件就会发送 WML。该控件还会检测设备并生成正确的标记。

1.3　ASP.NET 的开发环境

1.3.1　安装 Visual Studio 2012 for Web

1. 获取 Visual Studio

可以从 Microsoft 网站 www.microsoft.com/express/直接下载 Visual Studio 2012 的完整版进行安装，也可以只下载 Visual Studio Express 2012 for Web(VSEW)的免费版本。本书可以直接下载 Visual Studio Ultimate 2012with Blend 完整版进行安装。

2. Visual Studio 安装步骤

Visual Studio 的安装步骤非常简单，可以直接解压后双击 Setup.exe 进行安装，也可以通过加载至虚拟光驱进行安装。具体步骤如下。

(1) 双击 Setup.exe 后，打开如图 1-7 所示的安装界面。设置安装路径与目录，然后选中【我同意许可条款和条件】复选框。

(2) 单击【下一步】按钮，在如图 1-8 所示的向导对话框中选择要安装的可选功能，如果都需要安装则可以选中【全选】复选框，安装向导会提示 C 盘和安装盘有多少可用空

间。设置好以后，单击【安装】按钮。

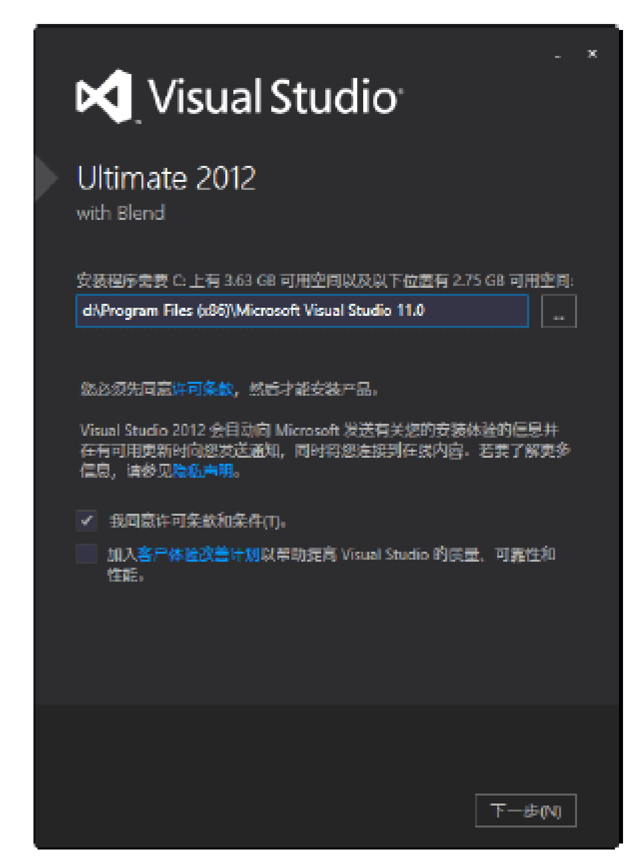


图 1-7　设置安装路径和同意许可条款



图 1-8　选择安装功能

(3) 向导进入安装提示界面，显示安装进度，如图 1-9 所示。安装完成后，显示【安装成功】界面，如图 1-10 所示。

(4) 这时候单击其中的【启动】按钮，将会显示一个启动对话框，告知用户本品使用权归属，如图 1-11 所示，然后进入 Visual Studio 2012 的工作界面，如图 1-12 所示。



图 1-9　安装进度界面

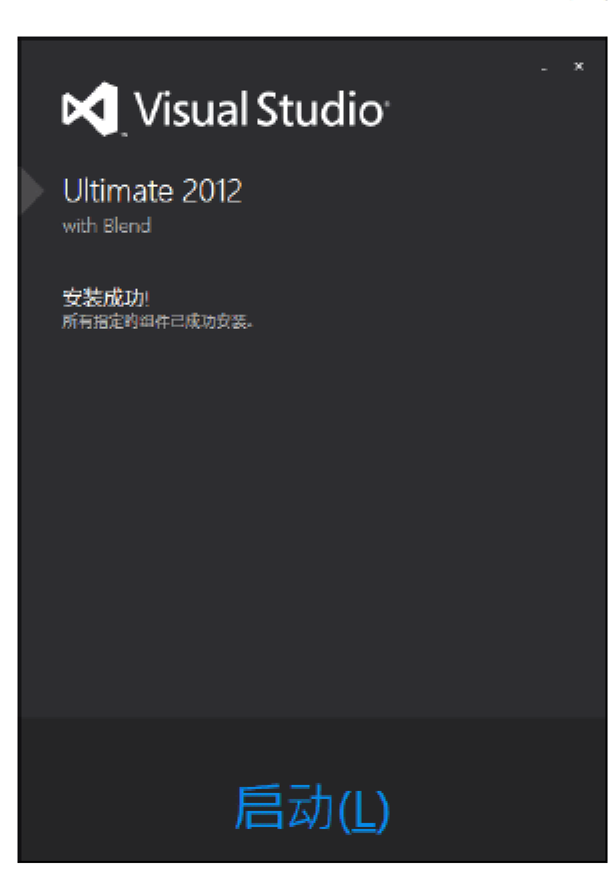


图 1-10　安装成功界面



图 1-11　启动界面



图 1-12　Visual Studio 2012 的工作界面

注意：

考虑到电脑资源和运行速度的问题，也可以安装 Visual Studio 2012 for Web 和 Microsoft SQL Express LocalDB Edition 11.0 作为开发环境。在安装的时候，通过 Microsoft Web Platform Installer(Web 平台安装程序)选择安装所需要的应用程序。

在 Express 的主页上，依次单击 Download 链接，直到打开提供了下载 Express 产品的页面，其中包括 VSEW。在这个页面上可以以 Web 安装方式下载 VSEW，如 Visual Studio Express 2012 for Web。

1.3.2　主开发区

Visual Studio 是目前为止构建 ASP.NET Web 页面使用最为广泛、功能最为丰富的集成开发环境(IDE)。缩略词 IDE 是指构建复杂 Web 应用程序所需的所有独立工具都集成在一个环境中。VS 不需要在文本编辑器中编写代码、在命令行中编译代码、在单独的应用程序中编写 HTML 和 CSS 以及在另一个应用程序中管理数据库，它允许在同一个环境中执行所有这些任务及更多其他的任务。由于不必一直进行工具转换，所以这样一来效率得到了提高，并且因为许多内置工具的工作方式都是一样的，所以人们能够更容易地学习 VS 的功能。

为了熟悉 VS 的界面中含有的众多工具，如图 1-13 所示。它显示了用 VS 创建第一个 Web 站点后所看到的屏幕，不过现在它突出显示了部分最重要的屏幕元素。如果已经熟悉了 Visual Studio 之前的版本，就可以跳过该部分，学习后面的其他知识。

如果安装的是 Visual Studio 以前的版本，屏幕可能会与图 1-12 有所不同，因为 Visual Studio 2012 能够导入老版本的设置。

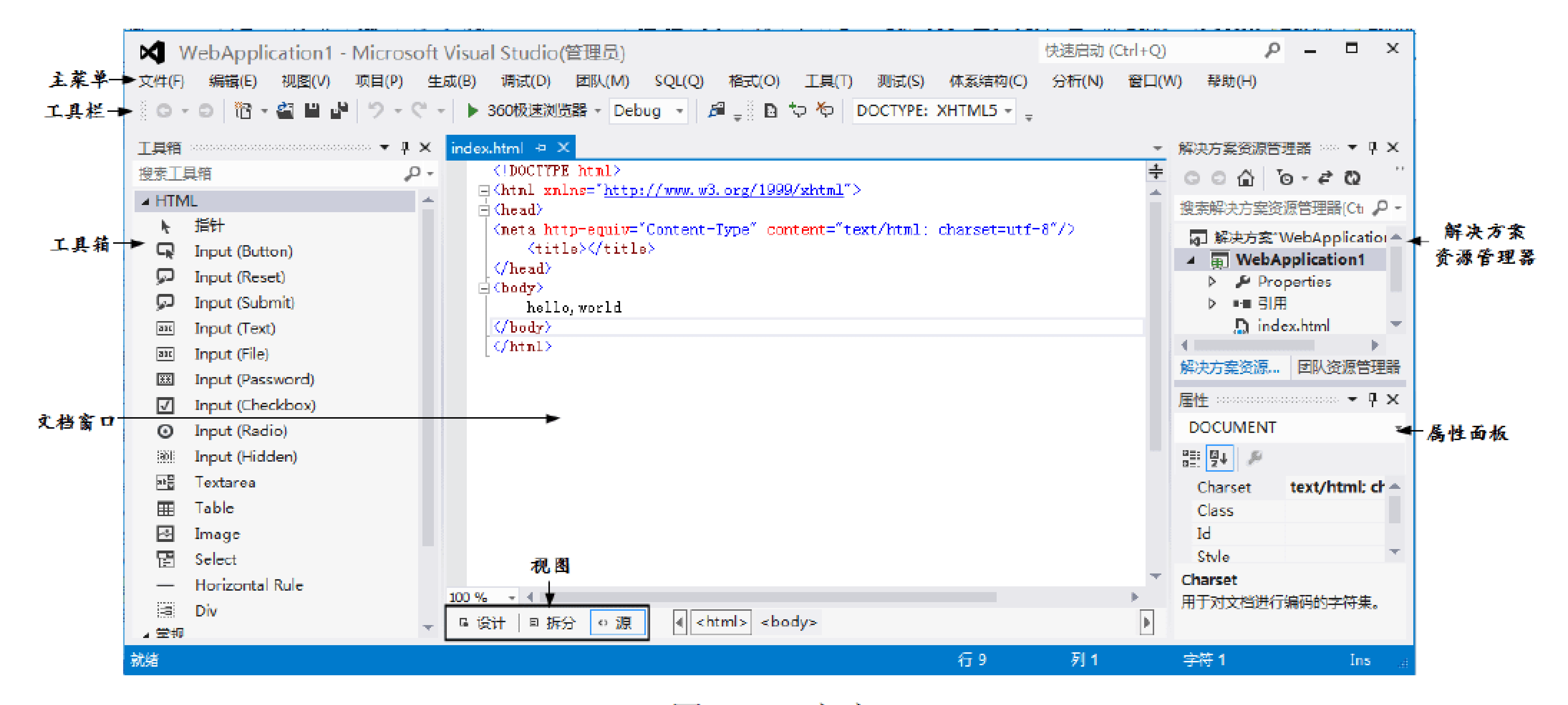


图 1-13　主窗口

1. 主菜单

在这个应用程序的上方，Windows 标题栏的下面，可以看到主菜单。此菜单栏中包含的菜单项用户一定很熟悉，因为在很多其他的 Windows 应用程序中都可以找到这些菜单

项，如【文件】、【编辑】、【帮助】菜单，以及一些 VS 中特有的菜单，如【调试】和【测试】菜单。该菜单栏可以根据执行的具体任务动态地改变，因此在使用应用程序的过程中开发者会发现某些菜单项有时出现、有时消失。可以通过使用【帮助】|【设置帮助首选项】菜单进行在线配置和离线帮助。离线帮助需要首先安装该应用程序，而在线帮助则需要连接到 Internet。

2. 工具栏区

在菜单的下方，可以看到工具栏区，在该区域显示了不同的工具栏，从而可以快速地访问 VS 中的大部分常见功能。在图 1-12 中，只启用了部分工具栏，但是在面向特定任务的场景中，VS 会随之打开很多可以使用的其他工具栏。有些工具栏会在执行需要特定工具栏出现的任务时自动地出现，但是也可以根据自己的喜好启用或禁用工具栏。要启用或禁用工具栏，只需要右击现有工具栏或菜单栏，从出现的菜单中选择工具栏即可。

3. 工具箱

在主屏幕的左边，可以看到折叠在 VS 边缘的【工具箱】选项卡。如果把鼠标指针悬停在该选项卡上，【工具箱】就会展开，这样就能看到它包含的内容。如果单击【工具箱】(或者有小钉图标的其他面板)右上角的小钉图标，它就会锁定在 IDE 上，保持打开状态。

与菜单栏和工具栏一样，【工具箱】会自动更新，以显示与正在执行的任务相关的内容。在编辑标准 ASPX 页面时，【工具箱】会显示可用于页面的许多控件。可以简单地从【工具箱】中拖动一个控件，然后把它放到希望在页面中出现的位置上。这些控件将在第 4 章详细讨论。注意每个【工具箱】类别还包含一个 Pointer 图标，但它不是一个控件。在 Visual Studio 的其他设计器(如 Win 窗体)中，这个图标用于退出控件绘制模式，但它在 ASP.NET 中没有什么用途。【工具箱】包含多个类别，其中的工具可以根据意愿展开和折叠，以便找到正确的工具。也可以重新排列列表中工具的顺序，从【工具箱】中添加和删除工具，甚至可以向其中添加自己的工具。关于定制 IDE 将在第 1.4 节讨论。

如果在屏幕上看不到【工具箱】，则可以按 Ctrl+Alt+X 组合键打开它，或者从【视图】菜单中选择【工具箱】命令来打开。

4. 解决方案资源管理器

在屏幕右边，可以看到【解决方案资源管理器】。【解决方案资源管理器】是一个重要窗口，因为它提供了组成 Web 站点的文件概览。【解决方案资源管理器】没有把所有文件都放在一个大文件夹中，而是将文件存储在单独的文件夹中，创建了一个有逻辑且有组织的站点结构。可以用【解决方案资源管理器】向站点中添加新文件，使用拖动功能或者剪切粘贴功能来移动现有文件，从项目中重命名文件以及删除文件等。【解决方案资源管理器】的大部分功能都隐藏在它的右击菜单中，该菜单根据在浏览器窗口中右击的项目而改变。

在【解决方案资源管理器】的上方有一个小工具栏，可以用来快速访问与 Web 站点相关的一些功能：包括刷新【解决方案资源管理器】窗口，嵌套相关文件的选项，以及用来

复制和配置 Web 站点的两个按钮。

可以通过从主菜单中选择【视图】|【解决方案资源管理器】或者按 Ctrl+Alt+L 组合键来访问【解决方案资源管理器】。

5. 服务器资源管理器

这个窗口一般默认不打开，打开以后默认显示在图 1-13 所示的【工具箱】右侧，通过它可以操作服务器资源，如数据库资源。如果使用的是 Visual Studio for Web，那么这个窗口就称为数据库源管理器，它可能位于屏幕右侧。

要访问【服务器资源管理器】，可以选择【视图】|【服务器资源管理器】，或者按下组合键 Ctrl+Alt+S。

6. 【属性】面板

用【属性】面板可以查看和编辑 Visual Studio 中的许多项目的属性，包括【解决方案资源管理器】中的文件、Web 页面上的控件、页面本身的属性及其他更多内容。这个窗口会不断地更新，以反映选中的项。按 F4 键可以快速打开【属性】面板。这个快捷键还可以用来强制【属性】面板显示选中项的详细信息。

7. 文档窗口

应用程序中间的文档窗口是主要区域。大部分动作都在这里发生。可以用文档窗口来操作很多不同的文档格式，包括 ASPX 和 HTML 文件、CSS 和 JavaScript 文件、VB 和 C# 的代码文件、XML 和文本文件，甚至图像文件。此外，用这个窗口还可以管理数据库、创建站点的副本，并在内置的微型浏览器中浏览页面等。

默认情况下，文档窗口是一个带选项卡的窗口，这意味着它能驻留多个文档，各个文档通过选项卡用窗口上方显示的文件名进行区分。各选项卡的右击菜单中包含使用该文件的一些有用的快捷键，包括保存与关闭文件，以及在 Windows 资源管理器中打开该文件的父文件夹。

要在文档之间进行切换，可以按 Ctrl+Tab 组合键，或者单击要查看的文档的标签，或者单击文档窗口右上角的下拉箭头，该文档窗口邻近【解决方案资源管理器】，如图 1-13 所示。单击下拉箭头会显示出一个打开文档的列表，因此可以轻而易举地从中选择要打开的文档。

切换文档的另一种方式是按下 Ctrl+Tab 组合键，然后按住 Ctrl 键。在弹出的窗口中，可以在右手边的那一栏中选择要使用的文档。然后可以在打开文档的列表中向上或向下移动光标。这样选择正确的文件就变得相当容易。

在同一个对话框中，可以看到一个包含所有活动工具窗口的列表。单击这个列表中的一个窗口，可以将它显示在屏幕上；如有必要，还能将它移到其他窗口的前面。

为了快速预览文档，而无须打开它，进行编辑，可以在【解决方案资源管理器】中单击要查看的文件，该文件会在其选项卡中处于预览模式，该选项卡停靠在右边一行，而不是左边放置打开文件的行上。

在图 1-13 所示的文档窗口下方可以看到 3 个按钮，分别是【设计】、【拆分】和【源】。在操作含有标记的文件(如 ASPX 和 HTML 页面)时，这些按钮会自动出现。它允许打开页面的【设计】视图(让开发者对页面在浏览器中的样子有个概念)、它的标记(HTML 和其他标记)，或者同时打开这两者。现在重要的是，知道可以通过单击相应的按钮在【设计】视图、【拆分】视图和【源】视图之间切换即可。

8. 起始页

每次启动 VS 时，【起始页】都要加载到文档窗口中。有了【起始页】，就可以快速地创建新的 Web 站点或者打开现有站点和其他项目。【起始页】还提供了一些有关 Web 开发的新闻和信息的链接。为了再次打开【起始页】，可以选择【视图】|【起始页】。

下面将来练习创建一个只有一个页面的新 Web 站点，该页面包含若干个 ASP.NET 服务器控件。该练习将介绍如何使用文档窗口和【解决方案资源管理器】，以及如何使用【工具箱】和【属性】面板向页面中添加 ASP.NET 服务器控件并修改它们的外观。

(1) 启动 Visual Studio 2012 或 VSEW。

(2) 如果使用的是 Express 版本，则通过【工具】|【设置】菜单，打开专家设置中的开发人员配置，以便访问 VSEW 的全部功能集。

(3) 在【文件】菜单中选择【新建】|【网站】，打开【新建网站】对话框。

(4) 在该对话框中，先选择模板语言是 VB.NET 还是 C#，然后确保选中的是【ASP.NET 空网站】，另外，一定要从【Web 位置】下拉列表中选择【文件系统】。然后单击【确定】按钮创建一个新的 ASP.NET 站点。

(5) 接着，右击【解决方案资源管理器】中新的 Web 站点，从显示的快捷菜单中选择【添加】|【新建项】命令。

(6) 在出现的【添加新项】对话框中，单击选中【Web 窗体】，并输入 DemoPage 作为名称。当单击【添加】按钮时，它会自动添加扩展名 ASPX。可以允许对话框中的其他设置保留其默认值。该页面应该能在【源】视图中打开，显示默认的 HTML 标签，如<html>、<head>、<title>和<body>元素，这些元素是在创建新页面时由 VS 自动添加的。

(7) 单击文档窗口下方的【设计】按钮，将页面切换到【设计】视图。

(8) 如果【工具箱】还没有打开，则按 Ctrl+Alt+X 组合键打开它，或者将鼠标悬停在【工具箱】选项卡上来显示它，然后单击锁定图标使【工具箱】一直可见。从【工具箱】的【标准】类别中将一个 TextBox 和一个 Button 拖动到页面【设计】视图中的阴影区域内。最后应当看到一个如图 1-14 所示的页面。

(9) 右击【设计】视图中的 Button 按钮并选择【属性】命令。在【属性】面板中，定位到【外观】类别下面的 Text 属性(如图 1-15 所示)，并将它从 Button 改为“提交信息”。一旦按下了 Tab 键或者在【属性】面板之外的某处单击，页面的【设计】视图就会更新，并在按钮上显示新文本。

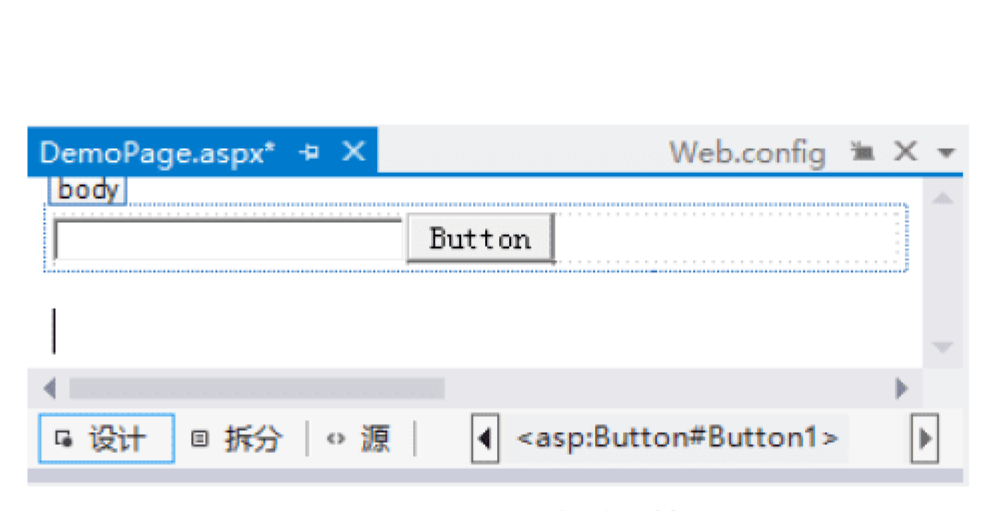


图 1-14　添加控件

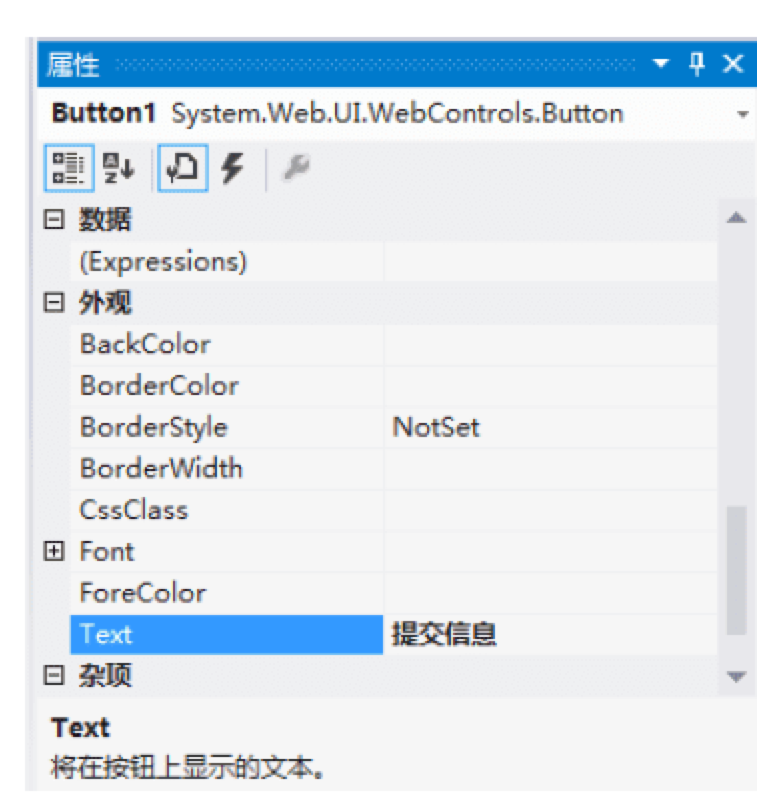


图 1-15　设置属性

(10) 按 Ctrl+F5 组合键在默认浏览器中打开该页面。注意，没有必要显式地保存对页面的修改(不过用快捷键 Ctrl+S 经常保存会有好处的)。按 Ctrl+F5 组合键运行页面后，VS 就会自动保存对打开文档的所有修改。

(11) 在文本框中输入一些文本，然后单击这个按钮。注意，当重新加载页面后，文本仍然会显示在文本框中。如果没有显示，则很可能是因为还没有为这个按钮编写任何代码。

当从【工具箱】中拖动 Button 和 TextBox 到页面的【设计】视图中时，VS 会自动在【源】视图中添加相应的代码。类似地，当在【属性】面板中修改按钮的 Text 属性时，VS 会自动更新【源】视图中控件的标记。如果不使用【属性】面板，也可以在【源】视图中 Text 属性的引号之间直接输入文本。

修改了 Text 属性之后，页面在【源】视图中应包含如下代码。

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<asp:Button ID="Button1" runat="server" Text="提交信息" />
```

当按 Ctrl+F5 组合键查看浏览器中的页面时，Web 服务器会收到请求，页面则由 ASP.NET 运行库处理，为页面产生的最终的 HTML 将被发送到浏览器。

当输入一些文本并单击按钮时，就会重复同样的过程：Web 服务器接收请求，处理页面，将结果发送回浏览器。当单击该按钮时，就引发了一个回发(postback)，此时页面中的所有信息(如在文本框中输入的文本)都会被发送回服务器。ASP.NET 通过再次呈现页面来对回发作出响应。然而，这次它用发送到页面的值预先填充了控件，如 TextBox。

使用浏览器的【查看源代码】命令查看页面最终的 HTML 代码(如果已经关闭了，则按 Ctrl+F5 组合键从 VS 中重新运行页面)。代码应如下所示。

```
<input name="TextBox1" type="text" id="TextBox1" />
<input type="submit" name="Button1" value="提交信息" id="Button1" />
```

从前面的示例中可以看出，最终的 HTML 与原始 ASPX 标记有相当大的区别。

在 VSEW 中驻留的窗口和工具面板远远不止目前为止所提到的这些。下一节将简要介绍一些构建 ASP.NET Web 页面时最常用的窗口。如果使用的是专家设置模式，则这里提到的所有窗口都可以从 VS 或 VSEW 的主【视图】菜单中访问。

1.3.3　信息窗口

除了在启动 VS 时见到的默认窗口外，VS 中还有很多可用窗口。在本书的其余部分，将运用到这些窗口，现在要先介绍几个重要的窗口。开发者可以访问接下来讨论的【主视图】菜单中的所有窗口。

1. 错误列表

【错误列表】提供了一个列表，列出了当前因为某种原因在站点中被中断的内容，包括 ASPX 或 HTML 文件中的错误标记，以及 VB 或 C#文件中的编程错误。这个窗口甚至可以显示 XML 和 CSS 文件中的错误。这个错误列表显示了 3 类消息——错误、警告和消息，它们分别表示不同的问题严重程度。如图 1-16 所示的是 CSS 和 XHTML 有问题的页面的错误列表。



图 1-16　错误列表

2. 【输出】窗口

当使用【生成】菜单构建站点时，【输出】窗口会提示是否构建成功。如果构建失败，如存在编程错误，那么【输出】窗口就会指出构建失败的原因。在 Visual Studio 的商业版本中，【输出】窗口还用来输出其他信息，包括外部插件程序的状态。

3. 【查找结果】窗口

当开始管理站点的内容时，VS 的【查找】和【替换】功能是非常有用的工具。在工作中经常需要替换当前文档甚至整个站点中的某些文本。在查找(Ctrl+Shift+F)和替换(Ctrl+Shift+H)时都会在【查找结果】窗口中输出它们的结果，如图 1-17 所示。

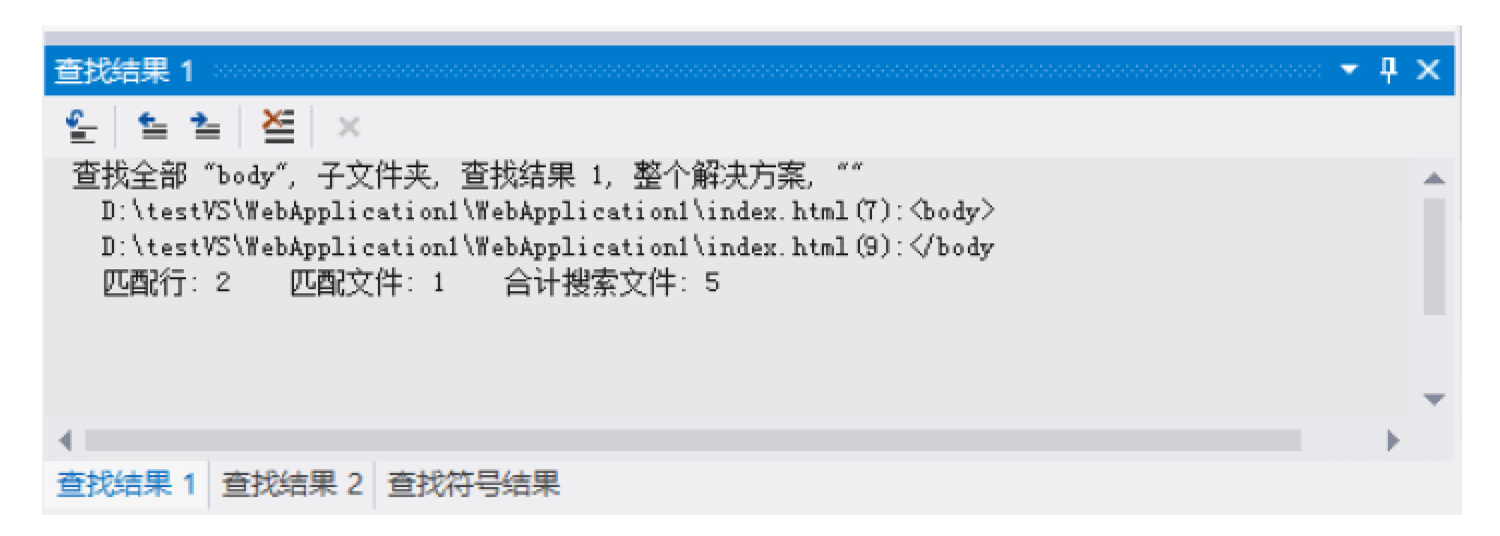


图 1-17　【查找结果】窗口

因为同时打开多个信息窗口可能会占用宝贵的屏幕空间，所以最好停靠(dock)它们。这样，一次就只能看到一个窗口，且仍然能够快速访问其他窗口。下一节将解释如何定制 IDE，包括如何停靠窗口。

1.4　定制 IDE

虽然 Visual Studio 的标准配置和它的工具窗口相当有用，但是开发者也可以根据自己的偏好定制 IDE。有时可能需要把窗口重新安排到容易找到的位置，有时可能希望打开其他经常用到的窗口。Visual Studio 是可以完全自定义的，而且 IDE 的任何细节都可以进行调整。下一节将介绍如何完成大多数常见的自定义任务。

1.4.1　重新排列窗口

可以在主 IDE 中通过拖动重新排列窗口。只须简单地按住窗口的标题栏或者它下面的选项卡，并沿着新位置的方向拖动即可。开始拖动窗口时，会看到 Visual Studio 提供的窗口将停在何处的可视化线索(如图 1-18 所示)。

如果按住指示器边上的 4 个方块指示器之一拖动窗口，窗口将停靠在现有窗口旁边。当放下它时，窗口就会出现在它的新位置上。如果将窗口放在指示器中间的方块上，窗口就会停靠在那个窗口的位置，并与之共享相同的屏幕空间。每个窗口都有自己的选项卡，这一点从图 1-18 下方的窗口上可以看出来。

除了在 IDE 中将窗口与其他窗口停靠在一起外，还可以使之成为浮动窗口。要把停靠窗口改为浮动窗口，可以将窗口从它的当前位置拖走，并放在 IDE 中的任何位置，不需要单击屏幕中的某个可视化线索。也可以从主菜单中选择【窗口】|【浮动】命令，或者右击窗口的标题栏，选择【浮动】命令，使其浮动。

要将浮动面板重新恢复到它原来的停靠位置，可以右击其标题栏，并选择【停靠】，或者从主菜单中选择【窗口】|【停靠】命令即可。

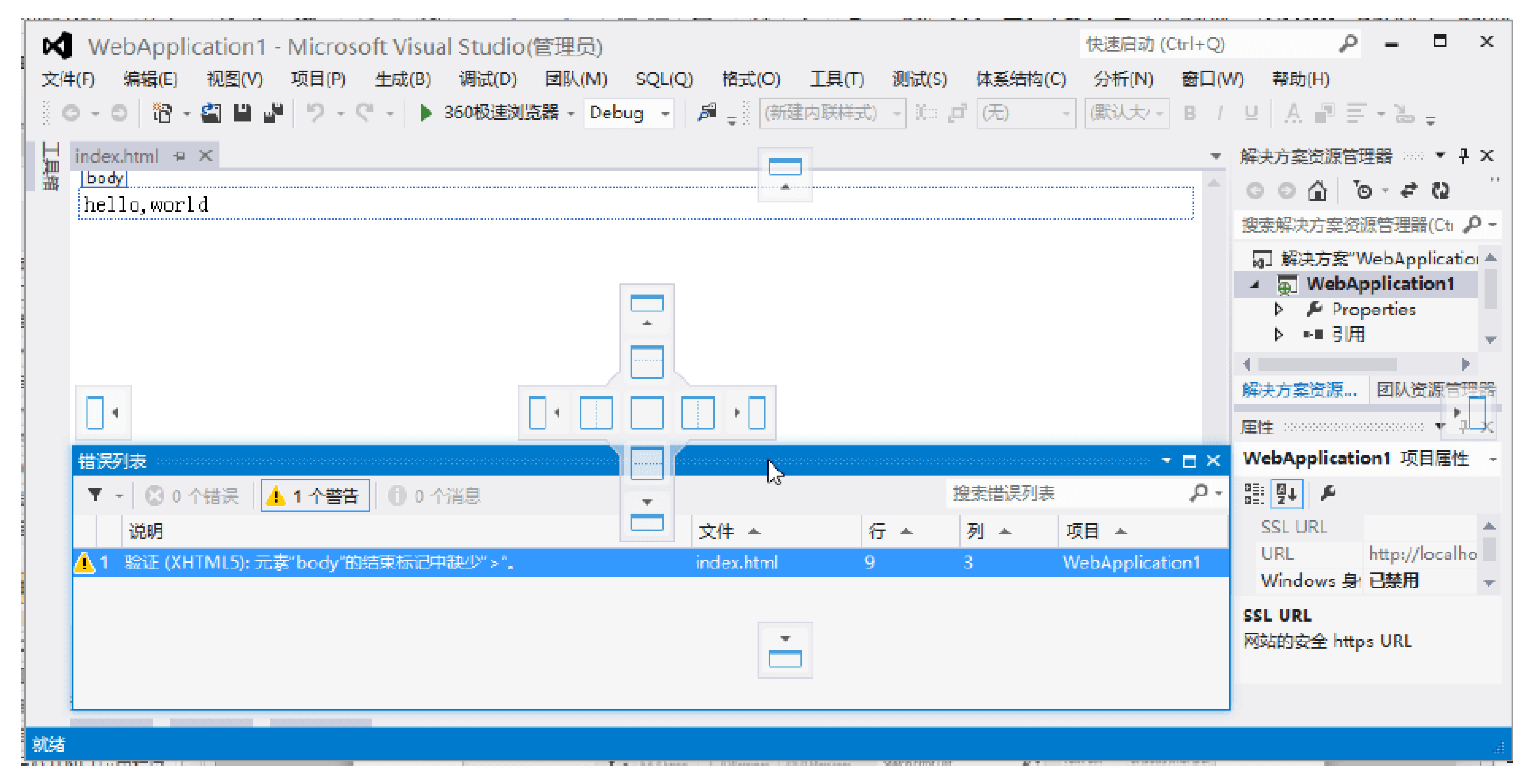


图 1-18　程序窗口

1.4.2　修改【工具箱】

【工具箱】也是可以修改的。默认情况下，【工具箱】按照字母顺序对控件进行分类，

但是也可以使用拖动工具对它们重新进行排列。为此，需要打开【工具箱】(按 Ctrl+Alt+X 组合键)，把一个控件(如【标准】类别下的 TextBox)拖动到一个不同的位置上。也可以通过右击它们并从上下文菜单中选择【删除】命令，把它们从【工具箱】中删除。不要担心控件会永久失去，因为可以从同样的菜单中选择【重置工具箱】命令来重置【工具箱】。

还可以向【工具箱】中添加自己的控件。这个功能最常用于代码段。只需要简单地突出显示文档窗口中的一些文本或代码，并把它拖动到【工具箱】中。然后可以右击该项目并选择【重命名项目】命令给它起一个更有意义的名称，以方便识别。

为了避免【工具箱】与自己的代码段混淆，可以为它们创建一个单独的类别。为此，可以从【工具箱】 的右击菜单中选择【添加选项卡】命令。输入一个名称，然后按下 Enter 键，【工具箱】选项卡就可以使用了。

下面练习打开和重新排列 VS IDE 中的窗口。

(1) 如果关闭了上文建立的 Web 站点，可以再次打开它，或者使用【文件】菜单新建一个站点。

(2) 从【视图】菜单中，选择【错误列表】命令，打开【错误列表】窗口。如果没有看到【错误列表】选项，则首先选择【工具】|【设置】|【导入和导出设置】。注意在默认情况下，它停靠在文档窗口的下方。

(3) 再次从【视图】菜单中，选择【任务列表】命令。默认情况下，它会停靠在与【错误列表】相同的空间中，两个窗口的选项卡彼此相邻。

(4) 单击【任务列表】选项卡，沿着文档窗口的方向将【任务列表】拖离它的位置。释放鼠标后，它就会显示为 IDE 中的浮动窗口。要恢复该窗口，则将它拖回到【错误列表】的中心方块上。要改变选项卡在选项卡组中出现的位置，把选项卡拖动到其他选项卡上，并在所需的位置处释放。

(5) 可以对 IDE 中默认可见的其他窗口或从【视图】菜单下找到的窗口重复上面的步骤。花一些时间来熟悉所有不同的窗口，并练习如何把它们排列在屏幕上。由于在本书的其余部分将会频繁地用到这些窗口，所以最好先熟悉它们的位置。

(6) 接下来，双击打开【解决方案资源管理器】中的 DemoPage 页面(或者如果创建了一个新的 Web 站点，则首先添加一个新的 ASPX 页面)。当该页面打开时，【工具箱】会自动可视。如果不可视，则按 Ctrl+Alt+X 组合键打开它。

(7) 右击【工具箱】并选择【添加选项卡】命令。输入 HTML Fragments 作为它的新名称，并按下 Enter 键。这样就向【工具箱】中添加了一个新类别，它的行为与所有其他类别一样。

(8) 当文档窗口在【源】视图中显示了 ASPX 页面后，则在起始<div>标记后直接输入<h1>。注意 Visual Studio 会自动地插入结束标记</h1>。最后在【源】视图中的代码如下所示。

```
<form id="form1" runat="server">
<div>
<h1></h1>
</div>
```

(9) 突出显示起始和结束<h1>标记，然后将【源】视图窗口中的选项拖动到在步骤(7)中创建的新【工具箱】选项卡上。该选项会显示为文本：<h1></h1>。

(10) 右击刚刚创建的【工具箱】项目，选择【重命名选项卡】命令，并输入 Heading 1 作为其名称。

(11) 再次把光标放到文档窗口中，按下 Ctrl+D 组合键后直接按 Ctrl+K 组合键，以便格式化文档窗口中的文档。另一种方法是从主菜单中选择【编辑】|【设置文档的格式】命令来格式化文档。这是根据在 Text Editor 选项对话框中设定的规则对文档进行格式化的。还可以对一些其他的文档类型进行格式化，包括 C#和 VB.NET 代码，以及 CSS 和 XML 文件。

从现在起，每当在【源】视图中的文档需要一个标题时，只须把光标放在文档窗口中希望标题出现的地方，然后双击【工具箱】中适当的标题即可。

下面来构建 Web 站点时经常需要的窗口。然后使用 IDE 的拖放功能将窗口的布局重新排列为个人偏爱的样子。

将一个 HTML 片段添加到【工具箱】的自定义选项卡中。当把任何标记拖动到【工具箱】中时，Visual Studio 都会为它创建一个含有选定标记的【工具箱】项。每当在页面中需要该标记的一个副本时，只要双击该项，或把它从【工具箱】中拖动到【源】视图窗口中即可。这样对于频繁用到的 HTML 片段可以节省很多时间。

最后，使用 Visual Studio 的文档格式化选项改变文档中代码的布局。这样有助于保持代码的有序性且使其更易于阅读。要完全改变代码的格式化方式，可以通过【工具】|【选项】命令打开【选项】对话框，然后扩展路径【文本编辑器】| HTML | 【格式设置】，并单击【标记专用选项】按钮，在打开的对话框中进行设置。

1.4.3　定制文档窗口

Visual Studio 使文本在文档窗口中的显示方式有极大的灵活性。可以修改一些属性，如字体大小、字体颜色，甚至是文本的背景色。要访问字体和颜色设置，可以选择【工具】|【选项】命令，然后选择【环境】 | 【字体和颜色】命令。

在文档窗口中，笔者喜欢定制选项卡的大小，它控制缩进代码时插入的空格数目。要修改 tab 的大小，选择【工具】|【选项】命令，然后在【文本编辑器】下方选择【所有语言】| 【制表符】选项。通常将【制表符大小】和【缩进大小】都设置为 2，其他设置不变。笔者还喜欢定制在 HTML 元素前后换行符的数目。【选项】窗口为此提供了完全的控制权：选择【文本编辑器】| HTML|【格式设置】命令，然后单击【标记专用选项】按钮，在左边的列表中可以选择一个标记，再通过右边的设置控制标记的格式化方式。【预览】框便于查看各种设置如何改变格式。

1.4.4　定制工具栏

工具栏可以通过 3 种方式定制：显示或隐藏内置工具栏、在现有工具栏上添加或删除按钮、使用常用的按钮创建自己的工具栏。

1. 启用与禁用工具栏

要禁用和启用已有的工具栏，可以右击任何现有工具栏或者菜单栏，然后从列表中选择合适的项。一旦显示了工具栏，就能用左边的拖动手柄把它拖动到工具栏区的新位置。

2. 编辑现有工具栏

如果感觉现有工具栏缺少了重要的按钮，或者工具栏中包含几乎用不到的按钮，则可以定制工具栏上的按钮。为此，可以右击任何工具栏或菜单栏，并选择【自定义】命令，然后切换到【命令】选项卡，并从【工具栏】下拉菜单中选择希望修改的工具栏。通过右边的【命令】按钮，可以添加新的命令并删除已有的命令，或者改变命令的顺序。

3. 创建自己的工具栏

如果想将常用的功能组合在一起，那么最好创建自己的工具栏。要创建一个新工具栏，用上一节介绍的方法打开定制窗口。单击【新建】按钮并输入工具栏的名称。然后切换到【命令】选项卡，用与修改现有工具栏一样的方式修改自己的工具栏。

1.4.5　定制键盘快捷键

很多开发人员喜欢修改的另一个设置是键盘快捷键。使用键盘快捷键是节省时间的好方法，因为它们允许用一个简单的键盘命令来执行任务，而不需要拿起鼠标选择菜单的正确项目。要修改键盘快捷键，选择【工具】|【选项】命令，展开【环境】，然后单击【键盘】选项，定位到命令列表中要修改快捷键的命令。由于这个列表中含有许多项目，因此可以通过输入命令的几个字母来过滤列表。例如，在【显示命令包括】字段中输入【打印】就会给出所有与打印相关的命令。

下一步，在【按快捷键】字段中，输入一个新的快捷键并单击【分配】按钮。Visual Studio允许为一个命令输入两个快捷键。例如，可以将命令【关闭所有文档】绑定到 Ctrl+K 组合键、Ctrl+O 组合键上。要执行这个命令，需要快速而连续地按下组合键。虽然两个快捷键看起来不太必要，但是它大大增加了可用快捷键的数目。

1.4.6　重置修改

如果觉得尝试了大量的自定义选项而弄乱了 Visual Studio，也不要担心。有多种办法可以把 Visual Studio 恢复到它以前的状态。

1. 重置 Window 布局

【重置 Windows 布局】命令可以从【窗口】菜单中访问，它用来将所有窗口重置为第一次启动 Visual Studio 时的位置。如果误放了太多窗口，最后得到的 IDE 比较混乱，那么这个命令就比较有用。

2. 重置【工具箱】

如果误删了【工具箱】中的项目，或者删除了整个选项卡，可以通过右击【工具箱】

并选择【重置工具箱】命令将【工具箱】重置为它的原始状态。在使用这个命令前需要慎重考虑，因为它也会删除所有的自定义代码段。

3. 重置所有设置

如果开发者做了一些练习，又进行了一些自定义设置，那么 IDE 现在很可能是这两种状态之一：要么看起来就是所希望的样子，要么看起来完全混乱了。对于后面这种情况，开发者需要先学习一下如何轻松地消除混乱状态。

要将所有的 Visual Studio 设置完全恢复到它们刚安装后的状态，请根据所使用的 VS 版本，选择【工具】|【导入和导出设置】命令。接下来，选择【重置所有设置】选项并单击【下一步】按钮。如果愿意，可以为现有设置创建一个备份；否则，就选择【取消】，仅重置所有设置即可。此时，会打开另一个界面，允许在众多的设置集合中进行选择。要选择专家设置或 Web 开发，因为这些选项提供了对本书中涉及的所有功能的访问。最后，单击【完成】按钮。这个动作会导致所有设置重置为它们的默认值，包括窗口布局、工具箱及工具箱定制、快捷键以及可能在 Visual Studio 选项对话框中修改了的所有内容。因此，只有当确实想要 Visual Studio 的一个全新配置时，才使用这个命令。

1.5　创建和运行 ASP.NET 应用程序

1.5.1　创建 ASP.NET 应用程序

配置好环境之后，就可以开始创建网站了。下面就来创建一个 ASP.NET 网站。

(1) 通过【开始】菜单启动 Visual Studio 2012，如图 1-19 所示。



图 1-19　VS 启动界面

(2) 选择【文件】|【新建】|【网站】命令，打开【新建网站】对话框，如图 1-20 所示。

(3) 在左边的【模板】中，选择 Visual C#(本书以 C#语言为主)。

(4) 在中间的模板列表框中，选择【ASP.NET 窗体网站】选项；在左下方的【Web 位置】下拉列表中选中【文件系统】；如果需要，可以改变 Web 站点在磁盘上的存储位置，单击【浏览】按钮，在计算机的硬盘驱动器上选择一个新位置即可。

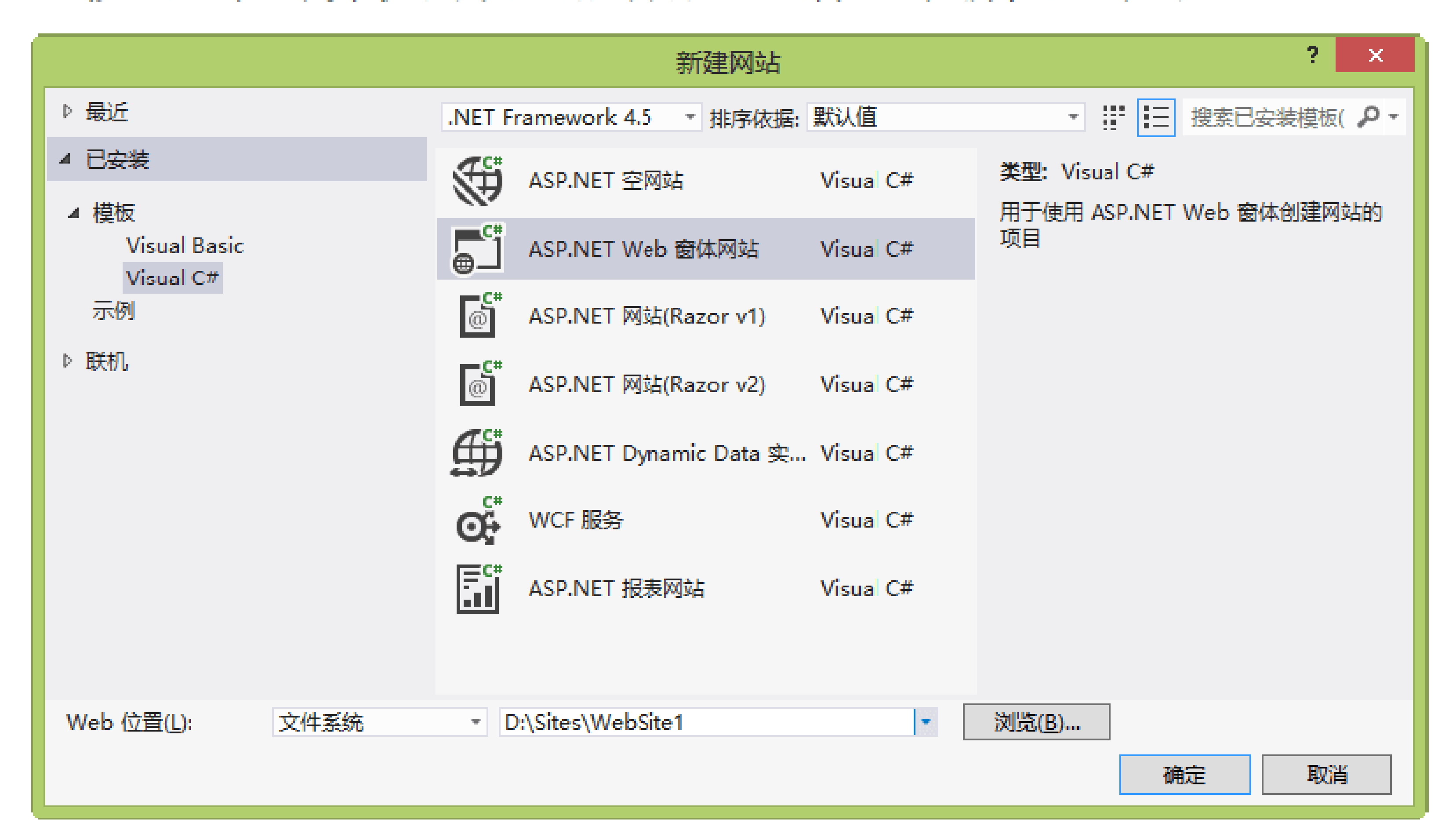


图 1-20　【新建网站】对话框

(5) 单击【确定】按钮，这时 VS 就创建了一个新的 Web 站点，其中包括如图 1-21 所示的文件和文件夹，它们可以用来开始创建 Web 站点。

(6) 双击文件 Default.aspx，删除 ID 为 FeaturedContent 的<asp:Content>代码块；删除 ID 为 BodyContent 的<asp:Content>代码块中的所有代码，用下面突出显示的文本和代码进行替换。

```
    <asp:Content runat="server" ID="BodyContent"
ContentPlaceHolderID="MainContent">
    <h2>Hello World</h2>
    <p>Welcome to Beginning ASP.NET 4.5 on <%:
DateTime.Now.ToString() %></p>
    </asp:Content>
```

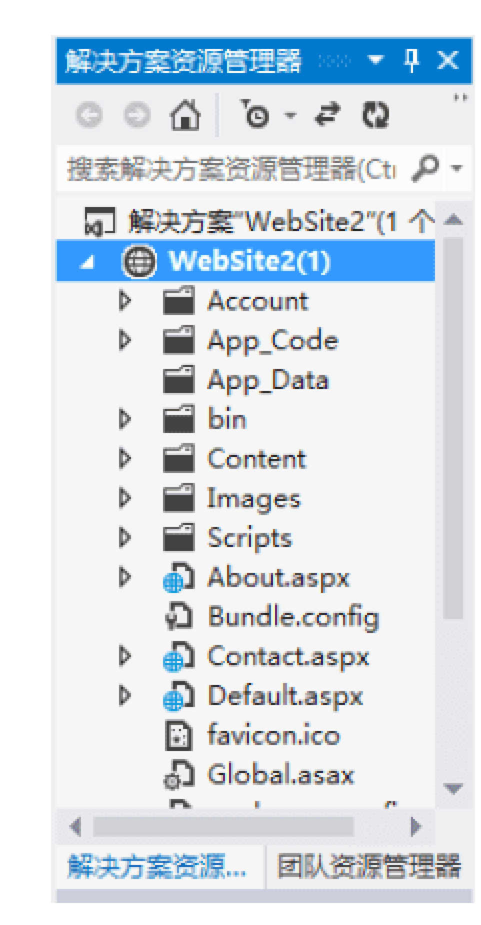


图 1-21　新建网站后生成的文件和文件夹

不用关心欢迎消息中使用尖括号(<>)的代码和百分号，它们称为标记，计算机自然就会知道它是如何工作的。虽然到目前为止，开发者可能还不熟悉这样的代码，但是不难猜出它是用来做什么的：输出今天的日期和时间。

1.5.2　运行 ASP.NET 应用程序

下面来运行所创建的应用程序。操作步骤如下。

(1) 按 Ctrl+F5 组合键，在默认的 Web 浏览器中打开页面，如图 1-22 所示。

如果开发者有一些 Visual Studio 经验，就可能习惯按下 F5 键。如果使用这个选项，站点就会在调试模式下打开，并显示一个对话框，询问是否要启用调试功能(可以安全地这么做)。



图 1-22　页面运行效果

如果在页面中没有看到日期和时间，或者收到了错误消息，则回顾一下欢迎消息中的代码。它以前尖括号(<)开头，后面跟着一个百分号和一个冒号，以一个百分号和另一个后尖括号(>)结束。还要确保输入和此处完全相同的代码，包括大小写也要一致。特别是使用的语言是 C#时这一点特别重要，因为该语言是区分大小写的。

(2) 注意，Windows 的任务栏中会出现一个表示 IIS Express 的小图标。如果看到另一个图标，就在 VS 中右击站点，选择 Use IIS Express。如果没有在任务栏中看到该图标，则可以单击 Windows 任务栏中其他图标旁边的箭头，并单击【自定义】选项。然后把 IIS Express System Tray 设置为 Show Icon and Notifications。这个图标属于内置的 Web 服务器 IIS Express。该 Web 服务器由 VS 自动启动，以响应对页面的请求。本书后面将会介绍 Web 服务器如何处理页面。

这就是用 Visual Studio 创建第一个 ASP.NET Web 站点的过程。虽然创建的这个 Web 站点相当简单，但是让 Default.aspx 页面显示在浏览器中的过程却没有那么简单。ASP.NET 页面(根据它的扩展名，也称为 ASPX 页面或 Web 窗体)本身并不能做太多的事。在浏览器能够显示它之前，需要一个 Web 服务器对它进行处理。这就是 VS 自动启动 IIS Express 来处理页面请求的原因。接下来，它会启动默认的 Web 浏览器并定向 Web 服务器的地址：http://localhost:57947/Default.aspx，不过每次启动 Web 服务器时，这个地址中的实际端口号可能都不同，因为该数字是 VS 随机选择的。

重要的是要意识到，在 VS 中修改的 ASPX 文件并不是在浏览器中最终显示的文件。

在 VS 中创建一个页面时，就向它添加了标记(markup)。ASPX 页面中的标记由以下内容组成： HTML、ASP.NET 服务器控件的代码、用 Visual Basic .NET 或 C#编写的代码等。

在浏览器中请求一个 ASPX 页面时，Web 服务器就会处理这个页面，执行它在文件中找到的所有服务器端代码，并有效地将 ASP.NET 标记转换为纯 HTML，然后发送给显示这个页面的浏览器。在上面的程序中，最终的 HTML 会引起浏览器显示当前日期和时间。HTML(HyperText Markup Language，即：超文本标记语言)是浏览器用来显示 Web 页面的语言。本章后面将会介绍 HTML，以及如何使用 HTML。

要查看最终的 HTML 与原始的 ASPX 页面有什么区别，请在浏览器中打开该页面的源代码。在大多数浏览器中，可以通过右击页面并选择【查看源代码】命令来打开源代码窗

口。这样还会打开一个默认的文本编辑器，用于显示该页面的 HTML 代码。

如果在完成上面的程序操作后已经关闭了浏览器，则也可以在 VS 中按 Ctrl+F5 组合键，再次打开该页面，并选择【查看源代码】命令。

在源代码中向下滚动，找到包含欢迎消息的代码行，注意标记之间不再是代码，而是实际的日期和时间。

```
<h2>Hello World</h2>
<p>Welcome to Beginning ASP.NET 4.5 on 2014/10/19 16:19:59</p>
```

当 Web 服务器处理页面时，它就从服务器中查询当前日期和时间，并把它们插入到要发送给浏览器的 HTML 中。根据 Windows 安装中的语言设置，会看到为适应 Windows 区域设置而进行格式化的不同日期和时间。

下一章将更详细地介绍 ASP.NET 的工作原理。

1.6　本章小结

本章介绍了很多重要的基础知识，使读者对 ASP.NET 4.5 和 Visual Studio 有了初步了解。首先从总体上简单介绍了 Microsoft .NET Framework 的历史，并特别介绍了 ASP.NET。然后说明了如何获得和安装 Visual Studio。Visual Studio 是创建 ASP.NET 4.5 Web 页面时使用最为广泛和最为通用的工具。为了更有效地使用这个工具，本章介绍了如何使用和定制 IDE。最后，通过实例演示了如何创建 ASP.NET 应用程序。

1.7　思考和练习

1. 简述静态网站和动态网站的区别。
2. 简述 Web 工作原理。
3. 简述 ASP.NET 开发环境 Visual Studio 的基本组成。
4. 简述通过 Visual Studio 创建网站的基本过程。

第2章　ASP.NET网页框架语言

HTML 是互联网上组建网站的最常用语言，但由于它本身固有的缺陷，已经不能满足动态网站的组建和互联网发展的需求。XML 是“万维网联盟”定义的受到广泛支持的行业标准，是新一代互联网的关键，但 XML 不兼容 HTML。因此，在现实中用 XML 取代 HTML 还有很长的路要走。XHTML 是基于 XML 的规范，并且是对 HTML 进行增强而形成的一种过渡语言。

本章对 XHTML 的概念、页面结构、语法规则和标记进行了详细的描述。通过本章的学习，读者能够掌握 XHTML 的基本概念，并学会使用 XHTML 编写 ASP.NET 网页。

本章的学习目标：

- 理解 HTML、XML 和 XHTML 的概念，以及三者之间的关系。
- 掌握动态网页的组成结构。
- 掌握 XHTML 的语法规则。
- 熟悉并使用 XHTML 标记。

2.1　Web 基本技术

互联网技术正处于日新月异的高速发展中，特别是目前互联网和移动网络的融合，更是加速了技术的发展。但是，无论是 Web 网站开发技术，还是目前流行的手机 APP 开发中，HTML、XML、XHTML 依然是最基本的技术，而 CSS、JS 技术均围绕它们而开展工作。本节将对这些最基本的 Web 技术进行简单介绍。

2.1.1　HTML

HTML(HyperText Markup Language)超文本标记语言，是制作页面文档的主要编辑语言。在任何操作系统下，只要有浏览器就可以运行 HTML 页面文档。作为一种标记语言，HTML 利用近 120 多种标记来标识网页的结构及超链接等信息，使页面文档在浏览器中展示出精彩纷呈的效果。HTML 只是建议 Web 浏览器应该如何显示和排列信息，并不能精确定义格式。因此，在不同的浏览器中显示的 HTML 文件效果会不同。

HTML 文件是一种纯文本文件，通常以.htm 或.html 作为文件扩展名。可以用各种类型的工具来创建或者处理 HTML 页面，如记事本、写字板、FrontPage、Dreamweaver 等。

HTML 由于它的简单易学，得到了广泛的使用，但是 HTML 存在着不可克服的缺陷。

首先，HTML 的标记是固定的。也就是说，HTML 不允许用户创建自己的标记。所以

HTML 很难做更复杂的事情，如无法描述矢量图形、科技符号和一些其他特殊显示效果。

其次，HTML 中标记的作用只是建议浏览器用何种方式显示数据。HTML 语言无法解释数据之间的关系，以及相关结构方面的信息。因此，不能适应日益增多的信息检索要求和存档要求。

通过上面的讨论可以看出，HTML 尽管十分简单方便，但当需要对一定量的数据进行复杂处理时，就力不从心了，而这正是 XML 可以大显身手的地方。

2.1.2　XML

HTML 是很成功的标记语言，目前很多网站都是由 HTML 语言制作的。HTML 语法要求比较松散，这对网页编写者来说，比较方便，但对计算机来说，语言的语法越松散，处理起来就越困难。传统的计算机能够处理松散的语法，但随着互联网的发展，对于许多新兴的连接到互联网的设备，如手机，解析网页语法的难度就比较大。于是，人们开始致力于构建另一种标记语言，使它既具有 HTML 的简单性，又具有强大的功能和可扩展性，XML 应运而生。

XML(Extensible Markup Language)可扩展标记语言，将网络上的文档规范化，并赋予标记一定的含义。同时，XML 不仅只是标记语言，它还提供了一个标准，用户可以利用这个标准定义新的标记语言，并为这个新的标记语言规定它所特有的一套标记。

XML 已经在文件配置、数据存储、基于 Web 的 B2B 交易、存储矢量图形和描述分子结构等众多方面得到广泛的应用。但是，由于目前的浏览器对 XML 的支持还不够完善，XML 在互联网上完全替代 HTML 还需要很长一段时间。

在由 HTML 向 XML 过渡阶段，国际万维网组织(W3C)在 HTML 基础上，按照 XML 格式制定了新的规范 XHTML 1.0，使网络编程人员只要通过简单地更改，就能将 HTML 转为 XHTML，从而为实现由 HTML 向 XML 的过渡找到桥梁。

2.1.3　XHTML

XML 虽然数据转换能力强大，完全可以替代 HTML，但面对成千上万已有的基于 HTML 语言设计的网站，直接采用 XML 还为时过早。因此，在 HTML 4.0 的基础上，用 XML 的规则对其进行扩展，得到了 XHTML(Extensible HyperText Markup Language)可扩展超文本标记语言。XHTML 是为了使 HTML 向 XML 顺利过渡而定义的标记语言。它以 HTML 为基础，采用 XML 严谨的语法结构。可以说，XHTML 结合了部分 XML 的强大功能及大多数 HTML 的简单特性，是一种增强了的 HTML。它的可扩展性和灵活性将适应未来网络应用的需求。越来越多的程序员开始利用 XHTML 设计网站结构，编写网页内容。

目前国际上在网站设计中推崇的 Web 标准就是基于 XHTML 的应用(即通常所说的 CSS+DIV)。本书所介绍的默认建立的网页即为 XHTML 格式的网页。

大部分的浏览器都可以正确地解析 XHTML，即使老版本的浏览器，也将 XHTML 作为 HTML 的一个子集。因此，几乎所有的网页浏览器在正确解析 HTML 的同时，都可以兼容 XHTML。

2.2　XHTML 的基本格式

XHTML 以 HTML 为基础，所以与 HTML 有很多相似之处。接下来介绍 ASP.NET 的页面文档结构和 XHTML 的语法规则。

2.2.1　ASP.NET 的文档结构

这里以一个页面 welcome.aspx 为例，来说明 ASP.NET 的文档结构。

welcome.aspx 的 XHTML 代码如下。

```
    <%@ Page Language="C#" AutoEventWireup="true" CodeFile="welcome.aspx.cs"
Inherits="welcome" %>
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
    <script runat="server">
    </script>
    <html xmlns="http://www.w3.org/1999/xhtml">
    <head runat="server">
        <title>无标题页</title>
    </head>
    <body>
        <form id="form1" runat="server">
        <div>
          <p>
            欢迎使用  Visual Web Develope
               
        </p>
        </div>
        </form>
     </body>
    </html>
```

从上面的代码可以看到，一个完整的 ASP.NET 页面文档是由指令、文档类型声明、代码声明、服务器代码、文本和 XHTML 标记等部分组成。

1. 指令

ASP.NET 页面通常包含一些指令，允许用户指定页面的属性和配置信息，对页面进行设置。指令指定的设置，不会出现在浏览器端。

在网页设计时，ASP.NET 提供“代码分离”技术，使开发者进行分工协作，分别进行网页界面代码设计和后台服务器运行代码设计。具体在实践中，将源代码放在扩展名为.aspx 文件中，将 Web 服务器运行代码放在另一个文件中。若此文件是由 C#编写的，则文件扩展名为.cs。这样做可以使前台 HTML 界面随着潮流不停地变化，而后台服务器端的代码可以稳定地实现业务处理。

.aspx 文件和.cs 文件的相互关联是由.aspx 文件中@page 指令连接的。例如，在本例中有如下指令。

```
<%@ Page Language="C#"  AutoEventWireup="true"  CodeFile="welcome.aspx.cs"
Inherits="welcome" %>
```

该指令说明编程语言为 C#，需要链接的 cs 文档为 welcome.aspx.cs。

2. 文档类型声明

DOCTYPE 为文档类型声明，指定本文档遵从的 DTD(Document Type Definition，即：文档类型定义)标准，同时指定了文档中的 XHTML 版本，可以和哪些验证工具一起使用等信息，以保证此文档与 Web 标准一致。

文档类型声明是每个网页文档必需的。如果网页文档中没有文档类型声明，浏览器就会采用默认的方式，即 W3C 推荐的 HTML 4.0 来处理此 HTML 文档。

本例的文档类型声明部分如下。

```
<！DOCTYPE  html  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">。
```

DOCTYPE 是 document type (文档类型)的缩写，W3C//DTD XHTML 1.0 Transitional 说明此文档符合 W3C 制定的 XHTML 1.0 规范，即声明此文档应该按照 XML 文档规范来配对所有标记。xhtml1-transitional.dtd 中的 dtd 是文档类型定义，包含了文档的规则，浏览器根据页面所定义的 dtd 来解释页面内的标识，并将其显示出来。

3. 代码声明

包含 ASP.NET 页面的所有应用逻辑和全局变量声明、子例程序和函数。页面的代码声明位于<script>…</script>标记中。

4. 服务器代码

大多数 ASP.NET 页面包含处理页面时在服务器上运行的代码。页面的代码位于 script 标记中，该标记中的开始标记包含 runat="server" 属性。

例如，本例中的<script runat="server">，说明页面运行时，ASP.NET 将此标记标识为服务器控件，并使其可用于服务器代码。

5. 文本和 XHTML 标记

页面的文本部分用XHTML标记来实现，这一部分结构应完全符合HTML的文件结构。在上面的例子中可以看到，一个最基本的 HTML 网页结构由 3 个部分构成。

```
<html>
    <head>
        <title>标题内容</title>
    </head>
```

```
        <body>
                主要内容
        </body>
</html>
```

- <html>…</html>：整个 HTML 文件的起止标记，其他 HTML 标记都要被放在这对标记之间。

在 HTML 代码中，仅有<html>…</html>，而在 XHTML 代码中使用了<html xmlns="http://www.w3.org/1999/xhtml">…</html>。其中的 xmlns 是 XHTML namespace 的缩写，即 XHTML 命名空间，用来声明网页内所用到的标记是属于哪个名称空间的。本例中，指定 HTML 的标记名称空间为 http://www.w3.org/1999/xhtml，这属于 XML 1.0 的写法。说明整个网页标记应符合 XHTML 规范。

- <head>…</head>：HTML 头部文件。

头部文件中包含页面传递给浏览器的信息。这些信息虽然作为一个单独的部分，不是网页的主体内容，但有时对于浏览器而言是很有用的。在头部文件中可以设置页面的标题、关键字、外部链接和脚本语言等内容。例如，如用<title>…</title>标记来设置网页的标题，用<script >…</script>标记来插入脚本等。

- <body>…</body>：文档内容部分。

<body>…</body>标记之间为页面文档的主体，用来放置页面的内容，是在浏览器中需要显示的内容。对一个最简单的网页来说，<body>…</body>标记符是必须使用的标记符。

2.2.2　XHTML 的语法规则

因为引入 XHTML 的目的是在 HTML 中使用 XML，所以 XHTML 的语法规则比 HTML 严格很多。具体有下列规则。

(1) UTF-8 之外的编码，文档必须具有 XML 声明。

当文档的字符编码是默认的 UTF-8 之外的编码时，编程人员必须在 XHTML 页面中添加一个 XML 声明，并指定代码。如下所示。

```
<? xml version="1.0"    encoding="iso-8859-1"?>
```

(2) 页面中的 HTML 标记之前必须使用 DOCTYPE 声明。

XHTML 1.0 提供了 3 种 DTD 声明供选择，DOCTYPE 声明必须引用其中一种类型。

- Transitional(过渡型)：要求使用宽松的 DTD，这时可以使用符合 HTML 4.0 标准的标记，但是必须符合 XHTML 的语法。这是 ASP.NET 所采用的默认文档类型定义。声明代码为如下。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

- Strict(严格型)：要求使用严格的 DTD，这时不能使用任何表现层的标记和属性，

如标记
。声明代码如下。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

- Frameset(框架型)：专门针对框架页面设计使用的 DTD，如果在网页中包含有框架，则需要采用这种 DTD。声明代码如下。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

(3) 页面的 html 标记必须指定命名空间。

html 标记必须指定 XHTML 命名空间，即将 namespace 属性添加到 html 标记中，如例子中的<html xmlns="http://www.w3.org/1999/xhtml">…</html>。

(4) 文档必须包含完整的结构标记。

文档必须包含 head、title 和 body 结构标记。框架集文档必须包含 head、title 和 frameset 结构标记。

(5) 标记必须正确嵌套。

XHTML 要求有严谨的结构，文档中的所有标记必须按顺序正确嵌套。

<p>This is a <i> bad example.</p></i>是错误的。

<p>This is a <i> good example.</i></p>是正确的。

也就是说，一层一层的嵌套必须是严格对称。

(6) 标记必须成对使用，若是单独不成对的标记，在标记最后加/>结束。

是错误的。

是正确的。

(7) 所有标记名称和属性的名字都必须使用小写。

与 HTML 不同，XHTML 对大小写是敏感的，XHTML 要求所有的标记和属性的名字都必须使用小写。<title>和<TITLE>在 XHTML 是不同的标记。

(8) 属性值必须用引号""括起来。

在 HTML 中，不要求给属性值加引号，但是在 XHTML 中，属性值必须被加引号。例如，<height=80> 必须修改为：<height="80">。

特殊情况下，若用户需要在属性值里使用双引号，可以使用'表示，如下。

```
<alt="say&apos;hello&apos;">
```

(9) 属性不允许简写，每个属性必须赋值。

XHTML 规定所有属性都必须有一个值，没有值的就重复本身。如下所示。

```
<input type="checkbox" name="shirt" value="medium" checked>
```

必须修改如下。

```
<input type="checkbox" name="shirt" value="medium" checked="checked">
```

(10) 使用 id 替代 name 属性。

(11) 图片必须有说明文字。

每个图片标记必须有 ALT 说明文字。即必须对 img 和 area 标记应用文字说明 alt="说明"属性。如下所示。

```
<img src="fish.jpg" alt="big fish" />
```

(12) 不要在注释内容中使用--。

--只能发生在 XHTML 注释的开头和结束，也就是说，在内容中它们不再有效。例如下面的代码是无效的。

```
<!--这里是注释-----------这里是注释-->
```

可以用等号或者空格替换内部的虚线，如<!--这里是注释====这里是注释-->是正确的。

以上规则的使用是为了使代码有一个统一、唯一的标准，便于以后的数据再利用。为由 HTML 向 XML 过渡打下基础。

2.3　XHTML 标记、标记属性

标记(Tags)是指定界符(一对尖括号)和定界符括起来的文本，用来控制数据在网页中的编排方式，告诉应用程序(如浏览器)以何种格式表现标记之间的文字。当需要对网页某处内容的格式进行编排时，只要把相应的标记放置在该内容之前，浏览器就会以标记定义的方式显示网页的内容。学习 XHTML 语言的重点就是学习标记的使用。

标记控制文字显示的语法如下。

```
    <标记名称>
        需进行格式控制的文字
</标记名称>
```

在 XHTML 标记中，往往还可以通过设定一些属性，来描述标记的外观和行为方式，以及内在表现，以便对文字编排进行更细微的控制。几乎所有的标记都有自己的属性。例如 style="text-align:center"，其中，style 就是标记的属性，style 的值设置文本格式为居中对齐。

使用标记符有如下一些注意事项。

- 任何标记都用<和>括起来，一般情况下，标记是成对出现的。
- 标记名与<之间不能有空格。
- 某些标记要加上属性，而属性只能加于起始标记中。格式如下。

```
<标记名　属性名=属性值　 属性名=属性值 …> 网页内容 </标记名>
```

XHTML 文件支持很多种标记，不同的标记代表不同的含义。XHTML 常用的标记包

括主体标记、注释标记、分层标记、文本标记、列表标记、表格标记、图像标记和超链接标记等。

2.3.1　主体标记<body>…</body>

主体标记之间定义了网页所有的显示内容。网页默认的显示格式为：白色背景，12 像素黑色 Times New Roman 字体。

在 XHTML 中，<body>标记用属性 style 来设置样式，如设置字体的大小、颜色、页面的背景色和背景图等。格式如下。

```
<标记 style="样式 1：值 1；样式 2：值 2；……">
```

其中，样式与值用冒号分隔，如果 style 属性中包含多个样式，各个样式之间用分号隔开。

style 属性常用的样式有以下几种。

- background-color：设置网页的背景颜色，默认为白色背景。
- color：设置网页中字体的颜色，默认颜色为黑色。
- font-family：设置网页中字体的名称，如宋体、楷体、黑体等。
- font-size：设置网页中字体的大小。
- text-align：设置网页中文本的对齐方式，常用有 3 种不同的取值：left(左对齐，默认对齐方式)、right(右对齐)、center(居中对齐)。

例如：<body style="font-family:宋体;color:blue">，设置网页字体为宋体，字体的颜色为蓝色。

2.3.2　注释标记<!--注释内容-- >

浏览器会自动忽略注释标记中的文字(可以是单行也可以是多行)而不显示。注释标记常用在比较复杂或多人合作设计的页面中，为代码部分加上说明，方便日后修改，增加页面的可读性和可维护性。

2.3.3　分层标记<div>…</div>

分层标记用来排版大块的 XHTML 段落，为 XHTML 页面内大块(block-level)的内容提供结构和背景的标记。可用 style 属性，在其中加入许多其他样式，以实现对其中包含元素的版面设置。

div 标记除了可以作为文本编辑功能外，还可以用作容器标记，即将按钮、图片、文本框等各种标记放在 div 里面作为它的子对象元素处理。

2.3.4　文本和格式标记

网页中最常用的就是文字了，下面将详细讲解 XHTML 中对网页中的文字进行格式设计和排版的常用标记。

1. 标题字体大小标记 <hn>…</hn>

设定网页的标题格式。由大至小，有 6 种设置标题格式的标记：<h1>、<h2>、<h3>、<h4>、<h5>和<h6>。

2. 字体的加粗、斜体和下划线标记

- …标记：以加粗字的形式输出文本。
- <i>…</i>标记：以斜体字的形式输出文本。
- <u>…</u>标记：以下划线形式输出文本。

3. 段落标记

- <p>…</p>

段落标记<p>…</p>的作用是将标记之间的文本内容自动组成一个完整的段落。

- 预格式化标记<pre>…</pre>

预格式化标记<pre>…</pre>使标记之间的文本信息能够在浏览器中按照原格式毫无变化地输出。它可以使浏览器中显示的内容与代码中输入的文本信息格式完全一样。

4. 换行标记

<br / >用于添加一个回车换行，该标记没有结束标记，故在 XHTML 中以</>结束。在编写 XHTML 时，如果在文件中用回车键分开了某一段文字，当在浏览器中显示时，浏览器会忽略源代码中的换行，而并不会显示换行的效果。若要显示网页中的文字换行效果，必须在文件中使用
标记。

5. 画线标记<hr />

画线标记< hr />单独使用，可以实现段落的换行，并绘制一条水平直线，并在直线的上下两端留出一定的空间。可以使用 style 属性进行设置。其中，有以下几点说明。

- width 用于设置画线的长度，取值可以是以像素为单位的具体数值，也可以使用相对于其父标记宽度的百分比数值。
- Height 用于设置画线的粗细，单位是像素。

6. 文本居中标记<center>… </center>

文本居中标记用来将网页中 center 标记内的元素居中显示。

【例 2-1】建立 ASP.NET 页面，命名为 example2-1.aspx。

主体部分代码如下。

```
<body style="text-align:center;font-family:楷体_GB2312;color:blue">
        <!--设置整个页面的字体居中显示，字体为楷体，颜色为蓝色-->
    <form id="form1" runat="server">
    <div >
            设定标题格式示例:
          <h1>设定标题格式，此处用 h1 效果</h1>
```

```
                <h6>设定标题格式，此处用 h6 效果</h6>
                    <hr style ="width:70%;height:10px;color:Black" />
                    <!--画一条分割线，宽度为整个页面的 70%，宽度为 10 像素，颜色为黑色-->
                <p> 字体的特殊效果示例：</p>
                <b>粗体显示</b><br />
                <i>斜体显示</i><br />
                    <hr />
            </div>
        </form>
    </body>
    </html>
```

在浏览器中查看，运行结果如图 2-1 所示。



图 2-1　对页面使用文本标记

7．列表标记

使用列表标记为网页中的文本设置表格。包括无序列表标记和有序列表标记。

● 无序列表标记
…
和列表项标记
…

无序列表是指各个列表项目没有顺序，显示时，在各列表项前面显示特殊符号的缩排列表，语法格式如下。

```
<ul  style="list-style-type">
<li>列表项 1
<li>列表项 2
…
<li>列表项 n
</ul>
```

其中，list-style-type 可以有几种形式：默认形式 disc(实心圆)●、circle(空心圆)○和 square(实心方块)■。

有自动换行的作用，每个条目自动为一行。每一个<li>创建的项目可以使用 list-style-type 单独指定的项目符号。

● 有序列表标记
…
和列表项标记
…

有序列表是在各列表项前面显示数字或字母的缩排列表，有序列表显示时，会在每个条目前面加上一定形式的有规律的项目序号。语法格式如下。

```
<ol  style="list-style-type">
<li>列表项 1
<li>列表项 2
…
<li>列表项 n
```

```
</ol>
```

其中，list-style-type 可以设为：upper-alpha(大写英文)、lower-alpha(小写英文)、upper-roman(大写罗马数字)、lower-roman(小写罗马数字)和 decimal(十进制数字)等。默认的列表标识符为阿拉伯十进制数字。

同无序列表中一样，<li>有自动换行功能。而且每一个<li>创建的项目可以使用 list-style-type 单独指定的项目符号。

【例 2-2】建立 ASP.NET 页面，名称为 example2-2.aspx。

其主体部分 XHTML 代码如下。

```
<body>
    <form id="form1" runat="server">
    <div>
        家用电器
      <ul>
          <li>大家电</li>
          <li style ="list-style-type:disc">平板电视</li>
          <li style ="list-style-type:circle">洗衣机</li>
          <li style ="list-style-type:square">家庭影院</li>
      </ul>
        运动户外
      <ol >
          <li>运动服饰</li>
          <li style ="list-style-type:lower-roman">卫衣/套头衫</li>
          <li style ="list-style-type:lower-alpha">训练服</li>
          <li style ="list-style-type:upper-roman">运动背心</li>
      </ol>
    </div>
    </form>
</body>
```

在浏览器中运行，结果如图 2-2 所示。

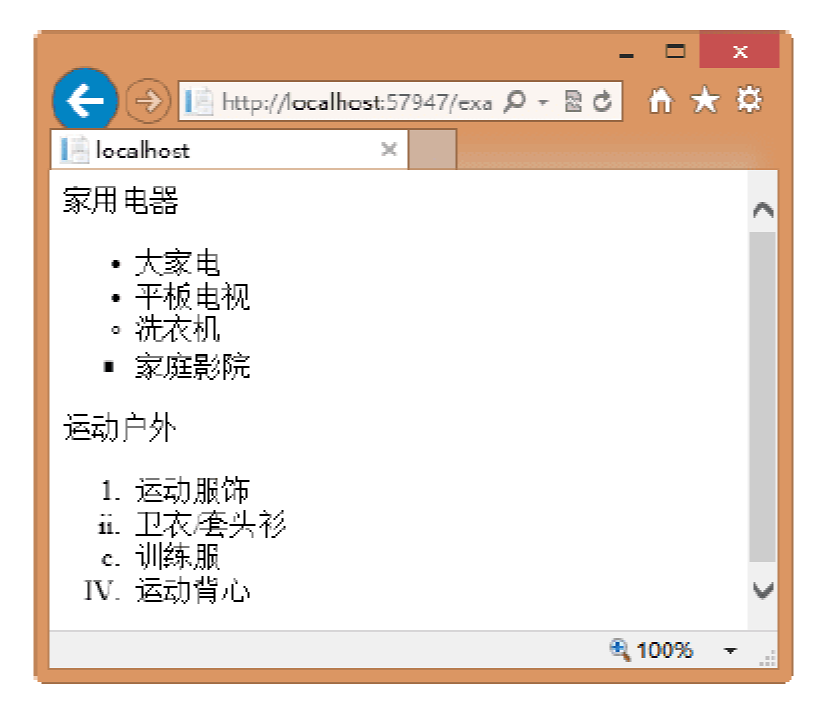


图 2-2　对页面使用列表标记

8. 空格标记

在 XHTML 中，直接输入多个空格，仅仅会被视为一个空格，而多个回车换行符也仅仅被浏览器解读为一个空格。为了能够显示多个空格，XHTML 保留了 HTML 中的空格标记 。一个 代表一个空格；多个 则代表相应的空格数。

2.3.5　表格标记

通过学习上面的文本标记，已可以对网页内容设置字体、段落、对齐方式等，但是由于浏览器的不同，并不能精确控制文本具体显示在网页的位置，而使用表格标记就可以对

网页中的各个元素的具体位置进行控制。因此，表格在网页设计中的定位功能是极其重要，同时也是在网页设计所有编辑方式中最灵活的。

表格由行与列组成，每一个基本表格单位称为单元格。单元格在表格中可以包含文本、图像、表单以及其他页面元素。

1. 表格标记<table>… </table>

表格标记用来声明表格，标志着一个表格的开始和结束，表格的所有定义都在这对标记范围内适用。<table>… </table>标记的常用属性有以下几种。

- align：设置表格在网页中的水平对齐方式，可选值 left、right、center。
- backGround：为表格指定背景图片。
- bgcolor：为表格设定背景色。
- border：设置表格边框厚度，如果此参数为 0，那么表格不显示边界。
- cellpadding：设置单元格中的数据与表格边线之间的间距，以像素为单位。
- cellspacing：设置各单元格之间的间距，以像素为单位。
- valign：设置表格在网页中的垂直对齐方式，可选值 top、middle、bottom。
- width：设置整个表格宽度。

2. 行起止标记<tr>… </tr>

此标记表明了表格一行的开始和结束，有以下属性。

- align：设置行中文本在单元格中的水平对齐方式，可选值 left、right、center。
- backGround：为这一行单元格指定背景图片。
- bgcolor：为这一行单元格设定背景色。

3. 单元格起始标记<td>… </td>

单元格起始标记用于设置表行中某个单元格的开始和结束。

【例 2-3】建立 ASP.NET 页面，名称为 example2-3.aspx。

其 XHTML 主体部分代码如下。

```
<body>
    <form id="form1" runat="server">
    <div>
      <table border="5">
        <tr align ="center" >
        <td bgcolor="red"> 第一行第一列，背景红色 </td>
        <td bgcolor="blue">第一行第二列，背景蓝色 </td>
        <td bgcolor="green">第一行第三列，背景绿色 </td>
        </tr>
             <!--以上设置第一行，文字居中-->
         <tr >
         <td align="left"> 第二行第一列，左对齐 </td>
         <td align="center">第二行第二列，居中 </td>
```

```
            <td align="right">第二行第三列，右对齐 </td>
            </tr>
           <!--以上设置第二行，文字居左-->
         </table>
       </div>
       </form>
   </body>
```

在浏览器中查看运行结果，如图 2-3 所示。

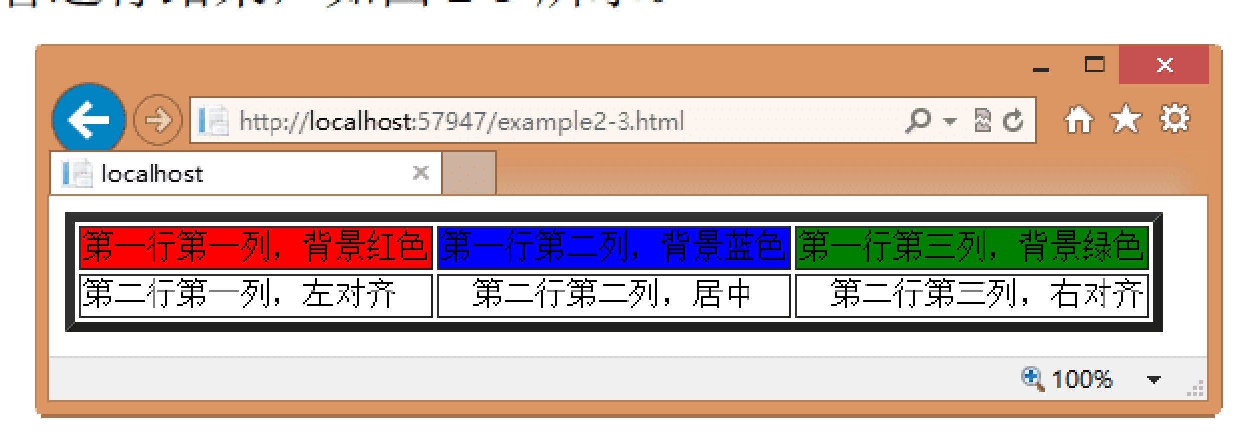


图 2-3　对页面使用表格标记

2.3.6　超链接标记<a>…

超链接是通过文字、图像等载体对文件进行链接，引导文件的阅读。互联网的魅力就在于可以通过超链接使任何一个网页，可以任意链接到世界任何角落的其他网页文件。超链接往往用不同的颜色或下划线与网页中的其他文字区别，在阅读文件时，用户通过单击超链接，能够随时查阅文件相关的详细信息。

1. 超链接命令的格式

超链接命令的格式如下。

```
   <a   href="URL"   id="设置锚点"   target="链接目标网页打开的窗口"> 锚点 </a>
```

(1) 锚点：实现链接的源点，通常当鼠标移动到锚标上会变成小手的形状，浏览者通过在锚标上单击就可以到达链接目标点。

(2) href 属性：设定要链接到的文件名称，为必选项。若文件与页面不在同一个目录，需要加上适当的路径，一般路径格式为 href="域名或 IP 地址/文件路径/文件名#锚点名称"。

(3) id 属性：用来定义页面内创建的锚点，在实现页面内部链接的时候使用。

(4) target 属性：设定链接目标网页所要显示的视窗，默认为在当前窗口打开链接目标。可选值为：_blank，_parent，_self，_top 及窗体名称。

- target="_blank"表示将链接的目标内容，在新的浏览器窗口中打开。
- target="_parent"表示将链接的目标内容，在父浏览器窗口中打开。
- target="_self"表示将链接的目标内容，在本浏览器窗口中打开(默认值)。
- target="_top"表示将链接的目标内容，在顶级浏览器窗口中打开。
- target="窗体名称"表示常用于框架或浮动框架中，将链接的目标内容，在“窗体名称”的框架窗体中打开，框架窗体名称已经事先在框架或浮动框架标记中命名。

如下所示。

```
<a href="http://www.taobao.com"   target="_selt" > 淘宝网</a>
```

这一段代码运行后，单击这个超链接，会在本窗口访问淘宝网。

2. 超链接的形式

XHTML 支持的超链接有以下几种形式：不同网页之间的跳转、链接至电子邮件、链接跳转到具体的锚点等。不同的超链接形式有不同的格式，具体如下。

● 链接到其他网页，基本格式如下。

```
<a   href="URL"> 锚点 </a>
```

在此处表示链接的是指定网页。运行时单击链接，转向另一个页面。

● 链接到图像上，基本格式如下。

```
<a   href="image_name.jpg" >锚点 </a>
```

运行时，单击超链接，跳转向一幅图片。

● 链接到电子邮件，基本格式如下。

```
<a   href="mailto:邮件地址"> 锚点 </a>
```

邮件地址形式为：name@site.come。

如下所示。

```
<a href="mailto: administrator@sohu.com" > 与搜狐网管理员联系</a>
```

运行后，单击超链接“与搜狐网管理员联系”，跳转到向管理员邮箱发信的页面。

● 页内链接：有的页面文本内容很多，浏览器打开页面往往从页面顶端开始显示，若用户需要的信息不在页面的起始部分，用户将费时费力地从上向下进行搜索。在此时，设置页内的链接是很有必要的。

实现页面内的链接时，需要先使用 id 属性定义一个锚点，格式如下。

```
<a   id="锚点名称">预被链接后显示的首部分</a>
```

然后再使用 href 属性指向该锚点，格式如下。

```
<a   href="#锚点名称"> </a>
```

#号表示链接目标与 a 标记属于同一个页面。

【例 2-4】建立 ASP.NET 页面，名称为 example2-4.aspx。

其主体部分代码如下。

```
<body>
    百度网站的超级链接：
    <a href="http:\\www.baidu.com"> 百度搜索</a><br />
</body>
```

在浏览器中查看，运行结果如图 2-4 所示。

图 2-4　对页面使用超链接标记

【例 2-5】建立 ASP.NET 页面，名称为 example2-5.aspx。

其主体部分代码如下。

```
<body>
    <form id="form1" runat="server">
    <div>
    第 2 章  XHTML 基础知识
    <ul>
        <li>2.1 web 基本技术</li>
        <li><a href="#html">2.1.1 HTML</a></li>
        <li><a href="#xml">2.1.2 XML</a></li>
        <li><a href="#xhtml">2.1.3 XHTML</a></li>
     </ul>
   <!--在网页头部设定指向锚点的超链接-->
        <p>
         <a id="html">2.1.1    HTML</a><br/>
         <!--创建锚点 html-->
         HTML(HyperText Markup Language)超文本标记语言，是制作网页文档的.....<br/>
        </p>
        <p>
         <a id="xml">2.1.2    XML </a><br/>
          <!--创建锚点 xml-->
         HTML 是很成功的标记语言，......<br/>
        </p>
        <p>
         <a id="xhtml">2.1.3    XHTML </a><br/>
         <!--创建锚点 xhtml-->
         XML 虽然数据转换能力强大，.....<br/>
        </p>
    </div>
    </form>
</body>
```

以上代码运行结果如图 2-5 所示。

2.3.7　图像标记<img />

Web 页面中的图像可以使网页更加生动、直观。常见的图像格式有 GIF、JPEG 和 PNG 等。其中，GIF 和 JPEG 格式能被大多数浏览器所支持。网页中的图像一般使用 72dpi 分辨率、RGB 色彩模式，在 XHTML 中使用<img />标记



图 2-5　对页面内部使用超链接标记

来向页面中插入图像。

图像标记语法格式如下。

```
<img  src="图像文件名"  [alt="提示文本"]  [border="边框宽度"]  [align="对齐方式"]
[width="宽度像素大小"]  [height="高度像素大小"]  alt="说明">
```

- src：这个属性是必需的，用来链接图像的来源。若图像文件与 XHTML 页面文件处于同一目录下，则只写文件名称；若图像文件与页面不在同一目录，需要加上合适的路径，相对路径和绝对路径均可。
- align：设置图像旁边文字的位置。可以控制文字出现在图片的上方、中间、底端、左侧和右侧。可选值为 top、middle、bottom、left 和 right，默认值为 bottom。
- alt：区别于 HTML，每个图片标记必须有 ALT 说明文字。若用户使用文字浏览器，由于浏览器不支持图像，这些文字会替代图像显示出来；若用户使用支持图像显示的浏览器，当鼠标移动至图像上时这些文字也会显现出来。

【例 2-6】将准备好的图片放入 WebSite2 的 images 文件夹，此处放入图片 f-b.jpg。建立 ASP.NET 页面，名称为 example2-6.aspx。

XHTML 主体部分代码如下。

```
<body>
    <form id="form1" runat="server">
    <div>
        <img src="/images/f-b.png" align="left" width="200" height="150" alt="冲锋衣" />  图片
左对齐，长 200 像素，宽 150 像素
    </div>
    <p>
    </p>
    <div align="center">
        <img / src="/images/f-b.png" align="middle"   width="100" height="100" alt="冲锋衣" />
图片居中，长 100 像素，宽 100 像素<br />
    </div>
    </form>
</body>
```

在浏览器中查看运行结果，如图 2-6 所示。



图 2-6　对页面使用图片标记

注意：当鼠标移至图片上时，会显示出 alt 属性的内容“冲锋衣”。

2.4　本章小结

HTML 是互联网上组建网站的最初常用的语言，但由于它本身固有的缺陷，已经不能满足动态网站的组建和互联网发展的需求。基于此，出现了 XML，它是“万维网联盟”定义的受到广泛支持的行业标准，是新一代互联网的关键，但 XML 不兼容 HTML。后来，人们将 XML 与 HTML 结合，建立了基于 XML 的 XHTML 规范。

本章对 XHTML 的概念、页面结构、语法规则和标记进行了详细的描述。通过本章的学习，读者能够掌握 XHTML 的基本概念，并会使用 XHTML 编写 ASP.NET 网页。

2.5　思考和练习

1. 简述 HTML、XML、XHTML 的基本概念。
2. 简述 XHTML 文档的构成。
3. 简单列举 XHTML 的标记，并简述其功能(至少 5 种)。
4. 创建一个 XHTML 页面 rumengling.aspx，输出结果如图 2-7 所示。



图 2-7　rumengling.aspx 输出结果

5. 用 XHTML 语言编写符合以下要求的页面：标题为 An example of image，在浏览器窗口中显示一个图像。图像的宽度为 200 像素，高度为 150 像素，边框宽度为 10 像素。

第3章　ASP.NET Web技术简介

本章是学习 ASP.NET 非常关键的一章，是从认识了解到使用 ASP.NET 的一个关键点。在使用 ASP.NET 进行程序开发时，其代码组织模式和页面的对象模型往往会贯穿整个开发过程，了解其基本运作方式有利于项目的顺利开发。

本章主要介绍 ASP.NET 的一些基础知识，学习和掌握这些知识是以后进行 ASP.NET 程序开发的基础和前提，本章正是围绕这些知识点进行介绍。首先介绍 ASP.NET 的页面框架和页面类，紧接着介绍了对 ASP.NET 各种程序文件进行存放的文件夹，接下来介绍 ASP.NET 的核心对象，主要包括 Response、Request、Application、Session、Server 等对象以及 ViewState 等，最后介绍了 ASP.NET 的配置文件 web.config 和全局文件 Global.asax。

本章的学习目标：

- 了解 ASP.NET 页面的运行机制和页面的生命周期。
- 了解 ASP.NET 所包含的文件夹以及各个文件夹的功能。
- 掌握定义、创建和使用常用内置对象的方法。
- 了解 Cookie 的基础知识以及设置和检索 Cookie 的方法。
- 掌握 ASP.NET 的配置管理方式。

3.1　ASP.NET 程序结构

3.1.1　ASP.NET 文件类型介绍

开发者在对程序进行设计时应该将特定类型的文件存放在特定文件夹中，以方便日后开发中的管理和操作。ASP.NET 保留了一些文件名称和文件夹名称，程序开发人员可以直接使用，并且还可以在应用程序中增加任意多个文件和文件夹，如图 3-1 所示。无须每次在给解决方案添加新文件时重新编译它们。ASP.NET 能自动、动态地预编译 ASP.NET 应用程序，并为应用程序定义好一个文件夹结构，利用这些定义好的文件夹，就可以自动编译代码，在整个应用程序中访问应用程序主题，并在需要时使用全局资源。下面介绍这些定义好的文件夹及它们的工作方式。



图 3-1　添加 ASP.NET 规定的特殊文件夹

3.1.2　ASP.NET 文件夹

1. App_Data 文件夹

App_Data 文件夹保存应用程序使用的数据库。它是一个集中存储应用程序所用数据库的地方。是 ASP.NET 为程序提供存储自身数据的默认位置，该文件夹内容不由 ASP.NET 处理。它通常以文件(诸如 Microsoft Access 或 Microsoft SQL Server 数据库、XML 文件、文本文件以及应用程序支持的任何其他文件)形式对数据进行存储。

注意：

默认 ASP.NET 账户被授予对文件夹的完全访问权限。如果碰巧要改变 ASP.NET 账户，一定要确保新账户被授予对该文件夹的读/写访问权。

2. App_Code 文件夹

App_Code 文件夹正好在 Web 应用程序根目录下，它存储所有应当作为应用程序的一部分动态编译的类文件。这些类文件自动链接到应用程序，而不需要在页面中添加任何显式指令或声明来创建依赖性。App_Code 文件夹中放置的类文件可以包含任何可识别的 ASP.NET 组件——自定义控件、辅助类、build 提供程序、业务类、自定义提供程序和 HTTP 处理程序等。

在开发时，对 App_Code 文件夹的更改会导致整个应用程序的重新编译。对于大型项目，这可能不受欢迎，而且很耗时。为此，鼓励大家将代码进行模块化处理到不同的类库中，按逻辑上相关的类集合进行组织。应用程序专用的辅助类大多应当放置在 App_Code 文件夹中。

App_Code 文件夹中存放的所有类文件应当使用相同的语言。如果类文件使用两种或多种语言编写，则必须创建特定语言的子目录，以包含用各种语言编写的类。一旦根据语言组织这些类文件，就要在 web.config 文件中为每个子目录添加设置，关于 web.config 文件将在后面章节进行介绍。

3. Bin 文件夹

Bin 文件夹包含应用程序所需的，用于控件、组件或者需要引用的任何其他代码的可部署程序集。该目录中存在的任何.dll 文件将自动地链接到应用程序。可以在 Bin 文件夹中存储编译的程序集，并且 Web 应用程序任意处的其他代码会自动引用该文件夹。典型的示例是如果为自定义类编译好了代码，那么就可以将编译后的程序集复制到 Web 应用程序的 Bin 文件夹中，这样所有页都可以使用这个类。

Bin 文件夹中的程序集无须注册。只要.dll 文件存在于 Bin 文件夹中，ASP.NET 就可以识别它。如果更改了.dll 文件，并将它的新版本写入到了 Bin 文件夹中，则 ASP.NET 会检测到更新，并对随后的新页请求使用新版本的.dll 文件。下面介绍一下 Bin 文件夹的安全性。

将编译后的程序集放入 Bin 文件夹中会带来安全风险。如果是自己编写和编译的代码，那么设计者是了解代码的功能的。但是，设计者必须像对待任何可执行代码一样来对待 Bin 文件夹中已编译的代码。在完成代码测试并确信已了解代码功能之前，要对已编译的代码保持谨慎的态度。Bin 文件夹中程序集的作用范围为当前应用程序。因此，它们无法访问当前 Web 应用程序之外的资源或调用当前 Web 应用程序之外的代码。此外，应该注意，在运行时，程序集的访问级别由本地计算机上指定的信任级别确定。

App_Code 文件夹和 Bin 文件夹是 ASP.NET 网站中的共享代码文件夹，如果 Web 应用程序要在多个页之间共享代码，就可以将代码保存在 Web 应用程序根目录下的这两个特殊文件夹中的某个文件夹中。当创建这些文件夹并在其中存储特定类型的文件时，ASP.NET 将使用特殊方式进行处理。

3.1.3　其他文件夹介绍

1. App_Themes 文件夹

主题是为站点上的每个页面提供统一外观和操作方式的一种新方法。通过 skin 文件、CSS 文件和站点上服务器控件使用的图像来实现主题功能。所有这些元素都可以构建一个主题，并存储在解决方案的 App_Themes 文件夹中。把这些元素存储在 App_Themes 文件夹中，就可以确保解决方案中的所有页面都利用该主题，并把其元素应用于控件和页面的标记。

2. App_GlobalResources 文件夹

资源文件是一些字符串表，当应用程序需要根据某些事情进行修改时，资源文件可用于这些应用程序的数据字典。可以在这个文件夹中添加程序集资源文件(.resx)，它们会动态编译，成为解决方案的一部分，提供给程序中的所有.aspx 页面使用。在使用 ASP.NET 1.0/1.1 时，必须使用 resgen.exe 工具，把资源文件编译为.dll 或.exe，才能在解决方案中使用。而在 ASP.NET 4.5 中，资源文件的处理就容易多了。除了字符串之外，还可以在资源文件中添加图像和其他文件。

3. App_LocalResources 文件夹

App_GlobalResources 文件夹用于合并可以在应用程序范围内使用的资源。如果对构造

应用程序范围内的资源不感兴趣，而对只能用于一个.aspx 页面的资源感兴趣，就可以使用 App_LocalResources 文件夹。可以把专用于页面的资源文件添加到 App_LocalResources 文件夹中，方法是构建.resx 文件名，如下所示。

- Default.aspx.resx
- Default.aspx.fi.resx
- Default.aspx.ja.resx
- Default.aspx.en-gb.resx

现在，可以从 App_LocalResources 文件夹的相应文件中检索在 Default.aspx 页面上使用的资源声明。如果没有找到匹配的资源，就默认使用 Default.aspx.resx 资源文件。

3.2　页面管理

ASP.NET 页面是带.aspx 扩展名的文本文件，可以被部署在 IIS 虚拟目录树之下。页面由代码和标签(tag)组成，它们在服务器上动态地编译和执行，为提出请求的客户端浏览器(或设备)生成显示内容。对于 Web 开发人员来说，如果想提高页面的运行效率，首先需要了解 ASP.NET 页面是如何组织运行的。

3.2.1　ASP.NET 页面代码模式

ASP.NET 的页面包含两个部分：一部分是可视化元素，包括标签、服务器控件以及一些静态文本等；另一部分是页面的程序逻辑，包括事件处理句柄和其他程序代码。ASP.NET 提供两种模式来组织页面元素和代码：一种是单一文件模式，另一种是后台代码模式。两种模式功能是一样的，可以在两种模式中使用同样的控件和代码，但使用的方式不同，接下来分别进行介绍。

1. 单一文件模式

在单一文件模式下，页面的标签和代码在同一个.aspx 文件中，程序代码包含在<script runat="server"></script>的服务器程序脚本代码块中间，并且代码中间可以实现对一些方法和属性以及其他代码的定义，只要在类文件中可以使用的都可以在此处进行定义。运行时，单一页面被视为继承 Page 类。

下面的代码示例演示一个单文件页，此页中包含一个 Button 控件和一个 Label 控件。突出显示的部分显示的是 script 块中 Button 控件的 Click 事件处理程序。

```
<%@ Page Language="C#" %>
<script runat="server">
void Button1_Click(Object sender, EventArgs e)
{
    Label1.Text = "Clicked at " + DateTime.Now.ToString();
}
```

```
</script>
<html>
<head>
  <title>Single-File Page Model</title>
</head>
<body>
  <form runat="server">
    <div>
       <asp:Label id="Label1"
         runat="server" Text="Label">
       </asp:Label>
       <br />
       <asp:Button id="Button1"
         runat="server"
         onclick="Button1_Click"
         Text="Button">
      </asp:Button>
    </div>
  </form>
</body>
</html>
```

script 块可以包含页所需的任意多的代码。代码可以包含页中控件的事件处理程序(如该示例所示)、方法、属性及通常在类文件中使用的任何其他代码。在运行时，单文件页被作为从 Page 类派生的类进行处理。该页不包含显式类声明。但编译器将生成将控件作为成员包含的新类。(并不是所有的控件都作为页成员公开；有些控件是其他控件的子控件。) 页中的代码成了该类的一部分；例如，创建的事件处理程序将成为派生的 Page 类的成员。

由于页中的所有代码都会成为 Page 类的成员。因此，代码中不能包含 using 语句。而应添加用于指定要导入的命名空间的 @ Import 指令，可能还需要添加一个对包含该命名空间的 DLL 的引用。

2. 后台代码模式

后台代码页面模式将可视化元素和程序代码分别放置在不同的文件中，如果使用 C#，则可视化页面元素为.aspx 文件，程序代码为.cs 文件，根据使用语言的不同，代码后缀也不同，这种模式也被称为代码隐藏模型。

ASP.NET 的代码隐藏模型，把一个程序文件分为一个.aspx 文件和一个对应的.aspx.cs 文件，前者是界面代码(主要用 html 编写)，后者是一些控制代码(主要用 c#编写)。.aspx 文件顶部的页面设置把两个文件联系在一块。在进行程序设计过程中，每一个控件都可以来触发事件。这些事件的代码单独在一个文件中，而网页的页面设计在单独的一个文件中，两个基本上是分离的，这样可以使代码文件更简洁。

在单一文件模型和代码隐藏模型之间，.aspx 页有两处差别。在代码隐藏模型中，不存在具有 runat="server" 特性的 script 块。如果要在页中编写客户端脚本，则该页可以包含不具有 runat="server" 特性的 script 块。第二个差别是，代码隐藏模型中的 @ Page 指令包含引用外部文件(SamplePage.aspx.vb 或 SamplePage.aspx.cs)和类的特性。这些特性

将.aspx 页链接至其代码。

在此示例中，用于标识代码隐藏文件的特性是 CodeFile。在 Web 应用程序项目中，该特性名为 CodBehind。

下面的代码示例演示一个与单文件页的示例包含相同 Click 事件处理程序的代码隐藏文件。

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
public partial class SamplePage : System.Web.UI.Page
{
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = "Clicked at " + DateTime.Now.ToString();
    }
}
```

代码隐藏文件包含默认命名空间中的完整类声明。但是，类是使用 partial 关键字进行声明的，这表明类并不整个包含于一个文件中。相反，编译器将读取.aspx 页以及它在@ Page 指令中引用的文件，然后将它们汇编成单个类。

3. 选择页模型

单一文件模型和代码隐藏页模型功能相同。在运行时，这两个模型以相同的方式执行，而且它们之间没有性能差异。因此，页模型的选择取决于其他因素。例如，要在应用程序中组织代码的方式、将页面设计与代码编写分开是否重要等。

(1) 单文件页的优点

通常，单文件模型适用于特定的页，在这些页中，代码主要由页中控件的事件处理程序组成。单文件页模型的优点包括以下几点。

- 在没有太多代码的页中，可以方便地将代码和标记保留在同一个文件中，这一点比代码隐藏模型的其他优点都重要。例如，由于可以在一个地方看到代码和标记，因此研究单文件页更容易。
- 因为只有一个文件，所以使用单文件模型编写的页更容易部署或发送给其他程序员。
- 由于文件之间没有相关性，因此如果使用的是 Visual Studio 之外的工具，则更容易对单文件页进行重命名。
- 因为页自包含于单个文件中，故而在源代码管理系统中管理文件稍微简单一些。

(2) 代码隐藏页的优点

代码隐藏页的优点使它们适用于包含大量代码或多个开发人员共同创建网站的 Web 应用程序。

代码隐藏模型的优点包括以下几点。

- 代码隐藏页可以清楚地分隔标记(用户界面)和代码。这一点很实用，可以在程序员编写代码的同时让设计人员处理标记。
- 代码并不会向仅使用页标记的页设计人员或其他人员公开。
- 代码可在多个页中重用。

3.2.2　页面的往返与处理机制

ASP.NET 网页是作为代码在服务器上运行的。因此，要使页面中的按钮或其他内容得到处理，必须将该信息提交到服务器。每次页面提交时，都会在服务器端运行其代码，然后把运行的结果呈现给用户。ASP.NET 页面的处理循环如下。

(1) 用户通过客户端浏览器请求页面，页面第一次运行，执行初步处理。程序员可以通过编程对页面进行初始化等操作。

(2) 执行的结果以标记的形式呈现给浏览器，浏览器对标记进行解释并显示。

(3) 用户输入信息或从可选项中进行选择，或者单击按钮。如果用户单击的是一个超链接而不是按钮，页面就会定位到另一个页面，而该页面不会被进一步处理。

(4) 页面发送到 Web 服务器，在 ASP.NET 中称此为“回发”，也就是说页面发送回其自身。例如，用户正在访问 Default.aspx 页面，则单击该页面上的某个按钮可以将该页面发送回服务器，发送的目的地还是 Default.aspx。

(5) 在 Web 服务器上，该页再次运行，并且使用用户输入或选择的信息。

(6) 服务器将执行操作后的页面以 HTML 或 XHTML 标记的形式发送到客户端的浏览器。

只要用户访问同一个页面，该循环就会继续。用户每次单击按钮时，页面中的信息就会发送到 Web 服务器，然后该页面再次运行。每个循环称为一次“往返行程”。由于页面处理发生在 Web 服务器上，因此页面可以执行的每个操作都需要一次到服务器的往返行程。

有时可能需要代码仅在首次请求页时执行，而不是每次回发时都执行，这时候可以使用 Page 对象的 IsPostBack 属性来避免对往返行程执行不必要的处理。

3.2.3　页面的生命周期

ASP.NET 页运行时，此页将经历一个生命周期，在生命周期中将执行一系列处理步骤。这些步骤包括初始化、实例化控件、还原和维护状态、运行事件处理程序代码以及进行呈现。了解页生命周期非常重要，因为这样做就能在生命周期的合适阶段编写代码，以达到预期效果。

与桌面应用程序中的窗体不同，ASP.NET 网页在使用窗体时不会启动或运行，并且仅当用户单击【关闭】按钮时才会卸载。这是由于 Web 具有断开连接的天性。浏览器从 Web 服务器请求页面时，浏览器和服务器相连的时间仅够处理请求。Web 服务器将页面呈现到浏览器之后，连接即终止。如果浏览器对同一 Web 服务器发出另一个请求，则即使是对同一个页面发出的，该请求仍会作为新请求来处理。

Web 这种断开连接的天性决定了 ASP.NET 页的运行方式。用户请求 ASP.NET 网页时，将创建该页的新实例。该页执行其处理，将标记呈现到浏览器，然后该页被丢弃。如果用户单击按钮以执行回发，将创建该页的一个新实例；该页执行其处理，然后再次被丢弃。这样，每个回发和往返行程都会导致生成该页的一个新实例。

如果要开发自定义控件，就必须熟悉页生命周期，以便正确进行控件初始化，使用视图状态数据填充控件属性以及运行控件行为代码。控件的生命周期基于页的生命周期，但是页引发许多需要在自定义控件中处理的事件。

一般来说，页要经历下表概述的各个阶段。除了页生命周期阶段以外，在请求前后还存在应用程序阶段，但是这些阶段并不特定于页。有关更多信息，请参考相关资料。

生命周期的某些部分仅当页处理为回发时才出现。对于回发，部分页回发过程中(当开发者使用 UpdatePanel 控件)与整页回发过程中的页生命周期是一样。

页生命周期阶段与说明如表 3-1 所示。

表 3-1　页面生命周期阶段与说明

阶　段	说　明
页请求	页请求发生在页生命周期开始之前。用户请求页时，ASP.NET 将确定是否需要分析和编译页(从而开始页的生命周期)，或者是否可以在不运行页的情况下发送页的缓存版本以进行响应
启动	在启动阶段，将设置页属性，如 Request 和 Response。在此阶段，页还将确定请求是回发请求还是新请求，并设置 IsPostBack 属性。页还设置 UICulture 属性
初始化	页初始化期间，可以使用页中的控件，并将设置每个控件的 UniqueID 属性。如果需要，还会向页应用于母版页和主题。如果当前请求是回发请求，则回发数据尚未加载，并且控件属性值尚未还原为视图状态中的值
加载	加载期间，如果当前请求是回发请求，则将使用从视图状态和控件状态恢复的信息加载控件属性
回发事件处理	如果请求是回发请求，则将调用控件事件处理程序。之后，将调用所有验证程序控件的 Validate 方法，此方法将设置各个验证程序控件和页的 IsValid 属性
呈现	在呈现之前，会针对该页和所有控件保存视图状态。在呈现阶段中，页会针对每个控件调用 Render 方法，它会提供一个文本编写器，用于将控件的输出写入页的 Response 属性的 OutputStream 对象中
卸载	完全呈现页并已将页发送至客户端、准备丢弃该页后，将引发 Unload 事件。此时，将卸载页属性(如 Response 和 Request)并执行清理

3.2.4　页面生命周期事件

对于 ASP.NET 开发者，理解 ASP.NET 的页面生命周期是非常重要的。主要是为了搞明白在哪里放置特定的方法和在何时设置各种页面属性。但是记忆和理解页面生命周期里提供的事件处理方法(method)非常困难。互联网上有很多关于页面生命周期内部机制的文章，所以本文只准备简单覆盖技术的基础部分，更主要的目的是给大家提供一个简单的记

忆页面生命周期的方法。

准确地记忆 ASP.NET 页面生命周期每一个阶段发生了什么事情是比较困难的，一种便于记忆的方法是根据各个阶段的名字组合出一个缩写。微软的文档给出的 ASP.NET 生命周期如下：

Page Request→Start→Page Initialization→Load→Validation→Postback event handling→Rendering→Unload

根据这个组合出一个缩写非常容易。既然 Page Request 技术上并不是页面生命周期的一部分(这个阶段仅仅标示我们是否开始一个页面周期或者从缓存加载一个页面)，为了方便，就不包括这一阶段。组合缩写如下：

- S – Start
- I – Initialize
- L – Load
- V – Validate
- E – Event Handling
- R – Render

这样就组合出一个缩写 SILVER，这个英文单词非常好记。当然，一定要记住页面生命周期的最后一个环节 unload 没有包括在里面。如果开发者觉得有必要，可以记忆为 SILVER-U 或者 SILVER-YOU，尽管有点破坏这个记忆法的完美性。现在，开发者可以非常容易就记住了页面生命周期，接着总结一下每一步都发生了什么，都有什么事件伴随着发生。

1. Start

在这个阶段，页面属性，如 Request、Response、IsPostBack 和 UICulture 被创建。最为一个开发人员，大部分时候在这个阶段不需要做任何事。如果需要调用或者重写(override)这一阶段的行为，可以使用 PreInit 方法创建或者重新创建动态控件，设置 master page 或者 theme 或者读取和设置 profile property 的值。要注意的一点是，如果是回传(postback)的页面请求，所有控件的值还没有从 view state 里还原，如果在这个阶段设置一个控件的值，这个值有可能在下面的阶段被重写并覆盖。

2. Initialize

这个阶段对于开发人员是很重要。在这个阶段，theme被应用，所有的控件都被设置了唯一的ID。开发人员在这个阶段可以调用Init、InitComplete和PreLoad 方法。微软关于这些方法使用的建议如下。

- Init：这个事件发生在所有控件被初始化并且皮肤设置也被应用后。使用这个事件来读取控件的初始化值。
- InitComplete：这个事件被 Page 对象触发，使用这个事件处理那些要求所有初始化工作都完成后才能做的事情。

- PreLoad：如果在页面或者控件进入 Load 事件前有什么要处理的，使用这个事件。Page 在触发这个事件后，Page 就会为自己和所有的控件加载 view state 并且处理所有 Request 中的 postback 数据。

3. Load

这个阶段可能是开发者使用得最多的一个阶段。在这个阶段，所有的控件被 viewstate 中信息填充并被加载，OnLoad 事件被触发。在这个阶段开发者可以为页面上所有的服务端控件设置属性，得到 query strings，建立数据库连接。

4. Validation

如果控件要求验证，验证会在这个阶段发生，这个时候开发者可以检查控件的 IsValid 属性。跟这个阶段关联的事件是 Validate，它有一个可以接受验证字符串群的重载方法(overload method)，这个重载方法执行特定控件群的验证。

5. Event Handling

所有服务器端控件的事件处理发生在这个阶段。也就是说 Click, Selected Index Changed 等这些事件会应用到服务器端控件，如果是页面请求是回传(postback)的话，这些事件的处理函数就会被控件触发。这个阶段可以使用的事件如下。

- LoadComplete：在这个阶段，页面上所有的控件加载完毕。
- PreRender：这里有几个重点，第一，页面对象(page object)会调用每一个控件的 EnsureChildControls 函数，并最终调用自己的。其次，所有具有 DataSourceID 的数据绑定控件都会调用自己的 DataBind 函数。要注意的一点是，PreRender 事件会发生在一个页面的每一个控件上。在这个事件的最后，页面和所有控件的 ViewState 被存储。
- SaveStateComplete：到这里，ViewState 已经存储完毕，如果有什么操作不需要修改控件但需要修改 ViewState 的，可以放在 SaveStateComplete 里面。

6. Render

渲染(Render)实际上不是一个事件，页面对象调用每一个控件的 Render 方法从而按顺序地输出控件的 HTML 代码。编写用户自定义控件的开发者对这个阶段最感兴趣了，因为输出用户自定义 HTML 代码的标准做法就是重写 Render 方法。如果控件是从 ASP.NET 服务器端控件继承来的，也许不需要重写 Render 方法，除非想呈现一个与用户控件默认行为不同的行为。

7. Unload

最后这个事件首先是被各个控件逐一触发，最后被页面触发。在这个时刻，所有的控件已经被渲染为输出流(output stream)并且无法被修改。这个阶段中，任何试图对 response stream 的操作都会引发异常。这个事件主要用于做一些清理工作，如关闭数据库连接和打开的文件或者登记事件记录等其他任务。

3.2.5　页面周期中的方法

下面列出 ASP.NET 页面生命周期中所有的方法，这些方法都可以被重写(override)，要注意的是这些方法有的会递归调用，有个会被页面中的内容重复调用，这个列表是按照页面加载时最通用的顺序依次排列的。

- Construct
- ProcessRequest
- FrameworkInitialize
- InitializeCulture
- If child controls are present:
- AddParsedSubObject
- CreateControlCollection
- AddedControl
- ResolveAdapter
- DeterminePostBackMode
- OnPreInit
- OnInit
- TrackViewState
- OnInitComplete
- OnPreLoad
- OnLoad
- OnLoadComplete
- EnsureChildControls
- CreateChildControls
- OnPreRender
- OnPreRenderComplete
- SaveViewState
- OnSaveStateComplete
- CreateHtmlTextWriter
- RenderControl
- Render
- RenderChildren
- VerifyRenderingInServerForm
- OnUnload
- Dispose

3.2.6　ASP.NET 页面指令

ASP.NET 页面中通常包含一些类似<%@....%>这样的代码，被称为页面指令。这些指令是允许相应指定的一些属性和配置信息的，就是 ASP.NET 用作处理页面的指令。常用的页面指令如下。

1. @ Assembly 指令

@Assembly 指令，即程序指令。用于在编译时将程序集链接到页面，这时程序员可以使用程序集公开的所有的类和方法。@Assembly 指令可以在 aspx 页面、.ascx 文件、.master 页面和.asax 文件中使用。语法格式，如下。

```
<%@Assembly Name="assName%">   //Name 表示指定编译页面时链接的程序集
<%@Assembly Src="pathName">    //Src 表示指定动态编译并链接到当前页面
```

对于@Assembly 指令中的 Name 和 Src 两个属性，它们是不能在同一个指令里同时存在。如果要同时存在的话，就必须在文件中要多包含一个@Assembly 指令。

在链接 Web 应用程序中 bin 目录的程序集时，将自动链接到应用程序中的 ASP.NET 文件。如果不想使用@Assembly 指令，只需要在 Web 应用程序中找到 Web.config 文件的<assemblies>节移除下面一行的代码，如下。

```
<add assembly="*">
```

2. @Control 指令

@Control指令，又称控制指令。该指令只能用于用户控件中，用户控件在带有.ascx扩展名的文件中进行定义。每个.ascx文件只能包含一条@Control指令，也只允许定义一个Language属性，每个控件只能使用一种语言。语法格式如下。

```
<%@Control attribute=”value”..%> //attribute 包含了所有的@Control 的属性
```

各属性如下。

- AutoEventWrieup：该控件的事件是否自动匹配，true 为自动，否则为 false。默认为 true。
- ClassName：一个字符串，用于指定需求在请求时经行动态编译的控件类名，此值可以是任何有效的类名，并且可以包括完整的命名空间，完全限定类名。如果没有设置此属性，则编译控件类名将基于控件的文件名。
- CodeFile：指定该控件代码的隐藏文件的路径。和 Inherits 属性一起使用，将隐藏代码源文件和用户控件的关联。该属性只对已经编译控件有效。
- Debug：是否应使用调试符号编译控件。如果是则 true，否则 false。由于该属性设置会影响性能，应该只在开发时，将该属性设置为 true。
- EnableTheming：指示控件是否使用主题，使用主题，则为 true，否则为 false。默认 true。

- EnableViewState：是否该控件请求维护视图状态。如果维护，则为 true，否则为 false。默认为 true。
- Inherits：定义供空间继承代码的隐藏类。是从 UserControl 类派生的任何类。和包含代码隐藏类源文件的路径 CodeFile 属性一起使用。
- Language：指定在编译控件中所有内联呈现(<% %>和<% = %>)和代码声明块时使用的语言。可以是任何.NET 支持的语言，VB、C#、JScript。每个控件，只能使用指定一种语言。
- Src：指定包含连接到控件的代码的源文件的路径。在所链接的源文件中，可以选择在类中或在代码声明块中包括控件的编译逻辑。

3. @Implements 指令

@Implements 指令，又称接口指令。指示当前的 ASP.NET 应用程序文件(网页，用户控件或母版页)实现指定的.NET Framework 接口。语法格式如下。

```
<%@Implements interface="ValidInterfaceName"%>
```

其中，interface 属性用来指定要在页或用户控件中实现的接口。

在 Web 窗体页中实现接口时，开发人员可以在代码声明块中<script>元素的开始标记和结束结束标记之间创建的事件、方法和属性，但是不能使用该指令在代码隐藏文件中实现接口。

4. @Import 指令

@Import 指令，即导入指令。该指令用于将空间显示导入到 ASP.NET 应用程序文件中，并且导入该命名空间的所有类和接口。导入的命名空间可以是 NET.Framework 类库的一部分，也可以是用户定义的命名空间的一部分。语法格式如下。

```
<%@Import namespace="value"%>
```

@Import 指令不能有多个 namespace 的属性，如果要导入多个命名空间，要使用多个@Import 指令。

3.3　ASP.NET 的常用内置对象

在 ASP.NET 里面，有几大常用内置对象，即 Response、Request、Application、Session、Server、ViewState、Cookie 等。下面将分别介绍一下这些对象的常用属性及方法。

3.3.1　Response 对象

Request 对象与 Response 对象就像一般程序语言里的 Input 及 Output 命令(或函数)，若要让 ASP.NET 程序能够接收来自前端用户的信息，或者想将信息传递给前端，都必须依

赖这两个对象。简言之，Request 对象掌管 ASP.NET 的 Input 功能，而 Response 对象则负责 Output 功能。

Response 对象实际是在执行 system.web 命名空间中的类 HttpResponse。CLR 会根据用户的请求信息建立一个 Response 对象，Response 将用于回应客户端浏览器，告诉浏览器回应内存的报头、服务器端的状态信息以及输出指定的内容。Response 的常用属性和方法如表 3-2 和表 3-3 所示。

表 3-2　Response 对象常用的属性列表

属　　性	说　　明
Buffer	获取或设置是否缓冲输出
Cache	获取缓存信息
CharSet	获取和设置输出流的 HTTP 字符集
ContentType	获取和设置输出流的 MIME 类型
Cookie	获取 Cookie 集合
Expires	获取和设置浏览器缓存超时时间
IsClientConnected	获取客户端是否和服务器连接
Status	设置返回给客户端的状态
StatusCode	获取和设置返回给客户端状态字符串
StatusDescription	获取和设置状态说明

表 3-3　Response 对象常用方法列表

方　　法	说　　明
AddHeader	添加 HTTP 头信息
AppendCookie	添加一个 Cookie
AppendHeader	添加 HTTP 头信息
AppendToLog	添加自定义信息到 IIS 日志中
BinaryWrite	以二进制的方式输出
Clear	清除输出缓存
Close	关闭和客户端的 Socket 连接
End	发送所有缓冲到客户端，并且停止执行页面
Flush	发送所有缓存到客户端
Redirect	重新定向 URL
SetCookie	更新一个已有的 Cookie
Write	输出信息
WriterFile	直接将指定文件写到输出流

下面举例说明该对象最常用的一些属性和方法。

- Response.ContentType()方法

ContentType 属性设定 Response 对象的输出内容类型，默认值为：text/html。如下所示。

```
Response.ContentType= "image/gif "
```

表示输出内容类型为 GIF 图形文件。

● Response.Expires=number

Expires 属性指定了在浏览器上缓冲存储的页距过期还有多少时间。如果用户在某个页过期之前又回到此页，就会显示缓冲区中的版本。若此属性在一页上设置了多次，则使用最短的时间。如指定了日期(Date)而未指定时间(Time)，页面在午夜失效。如指定了时间而未指定日期，则在脚本执行的当天那个时间失效。如下所示。

```
Response.ExpiresAbsolute=DateTime.Now
```

表示页面在当前时间失效，即马上过期。

● Response.Buffer=Flag

Buffer 属性表示是否对页面输出进行缓冲。如有缓冲，服务器在所有当前处理的页面的语句被处理之前不将 Response 送往客户端，除非有 Flush 或 End 方法被调用。参数 Flag 表示是否对页面输出进行缓冲。True 表示需要，False 表示不需要，默认值是 True。

● Response.Flush()方法

立即将缓冲区中页面输出。

● Response.Write()方法

Write 方法输出指定的文本内容。如下所示。

```
Response.Write("Hello World! ")
```

● Response.WriteFile()方法

直接将输出的内容写入一个文件中。如下所示。

```
Response.WriteFile("c:\\write.txt")
```

【例 3-1】演示 Response.Write()方法的使用。

(1) 新建一个名为 WebSite3 的网站，在其中添加一个名为 example3-1.aspx 的网页。

(2) 在 example3-1.aspx.cs 页面中添加下面代码。

```
protected void Page_Load(object sender, EventArgs e)
{
        Response.Write("Hello World!");
        Response.Write("<h2>Hello World!</h2>");
        Response.Write("<p style='color:#0000ff'>Hello World!</p>");
}
```

(3) 程序运行效果如图 3-2 所示。

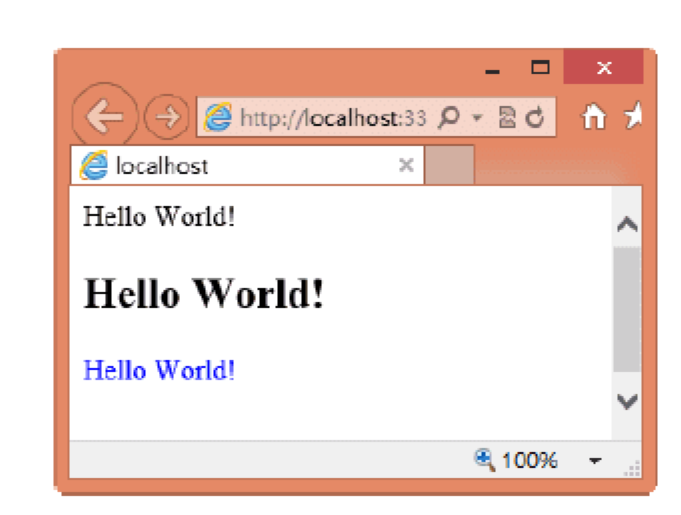


图 3-2　Response.aspx 运行效果

3.3.2　Request 对象

Request 对象是 ASP.NET 当中最有用的对象之一，它与 Response 对象一起使用，达到沟通客户端及服务器端的作用，使它们之间可以很简单地交换数据。Request 对象接收客户端通过表单或者 URL 地址串发

送来的变量，同时，也可以接收其他客户端的环境变量，如浏览器的基本情况、客户端的 IP 地址等。所有从前端浏览器通过 HTTP 通信协议送往后端 Web 服务器的数据，都是借助 Request 对象完成的。总而言之，Request 对象含有用户端最为丰富的数据。

Request 对象常用的属性和方法如表 3-4 和表 3-5 所示。

表 3-4　Request 对象常用属性列表

属　　性	说　　明
ApplicationPath	获得 ASP.NET 应用程序虚拟目录的根目录
Browser	获取和设置客户端浏览器的兼容性信息
ContentLength	客户端发送信息的字节数
ContentType	获取和设置请求的 MIME 类型
Cookies	获取客户端 Cookie
FilePath	当前请求的虚拟路径
Files	获取客户端上传的文件集合
Form	获取表单变量集合
Headers	获取 HTTP 头信息
HttpMethod	HTTP 数据传输方法，如 GET、POST
Path	获取当前请求的虚拟路径
PhysicalPath	获取请求的 URL 物理路径
QueryString	获取查询字符串集合
ServerVariables	获取服务器变量集合
TotalBytes	获取输入文件流的总大小
Url	获取当前请求的 URL
UrlReferrer	获取该请求的上一个页面
UserAgent	客户端浏览器信息
UserHostAddress	客户端 IP 地址
UserHostName	客户端 DNS 名称
UserLanguages	客户端语言

表 3-5　Request 对象方法列表

名　　称	说　　明
BinaryRead	以二进制方式读取指定字节的输入流
MapPath	映射虚拟路径到物理路径
SaveAs	保存 HTTP 请求到硬盘
ValidateInput	验证客户端的输入是否存在危险的数据

ASP.NET 是使用表单(Form)来实现用户数据提交的。对于 HTML 表单，可以使用 Get 方法或 Post 方法来实现数据提交。如果使用 Get 方法，就要使用 Request 对象的 QueryString 集合来得到相关的信息；如果使用 Post 方法，就要使用 Request 对象的 Form 集合来得到相关信息。下面分别讲解如何使用 Get 方法和 Post 方法。

1. Get 方法

使用 Get 方法进行数据提交需要使用 Request.QueryString 方法来取得传递的数据，传输的数据量比较小，一般限制在 2KB 左右，用户要提交的信息往往是作为查询字符串加在 URL 的后面传给接收程序。URL 是指 Uniform Resource Locator (统一资源定位器)，它的通常形式是类似 http://www.abc.com 形式的字符串。而查询字符串则是加在 URL 的后面，与前面的 URL 一起提交给服务器的字符串。例如：http://www.domain.com/test.aspx?name=myname& password=mypassword。

2. Post 方法

Post 方法是用 Request 对象的 Form 集合属性来得到用户提交的数据。使用该方法时，用户浏览器的地址栏中不会显示相关的查询字符串。因此，如果需要提交的数据很多时，应使用 Post 方法，因为它对数据的大小和长度没有什么限制。另外，如果使用 Web 表单而不是 HTML 表单的话，那么也只能使用 Post 方法来获取数据，因为这是 Web 表单默认使用的方法，无须在定义表单时说明。

与 Get 方法相比，Post 方法对数据的大小和长度没有什么限制，所以对于大量的数据提交，使用 Post 方法是十分有利的。另外，由于地址栏中不显示相关的查询字符串，那么使用 Post 方法就十分适合用来传递保密信息，如用户的账号和密码等。

【例 3-2】演示利用 Request 对象的 Form 集合来得到相关信息的方法。

(1) 在 WebSite4 网站中添加一个名为 example3-2-1.aspx 和名为 example3-2-2.aspx 的网页，在 example3-2-1.aspx 页面的<body>标签中添加如下代码。

```
<form id="form1" runat="server">
<div>
    <a href="example3-2-2.aspx?username=chen">Example</a>
</div>
</form>
```

(2) 在 example3-2-2.aspx.cs 页面中添加下面代码。

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write(Request.QueryString);
      Response.Write("<br/>");
    Response.Write(Request.QueryString["username"]);
}
```

(3) 程序运行效果如图 3-3 和图 3-4 所示。

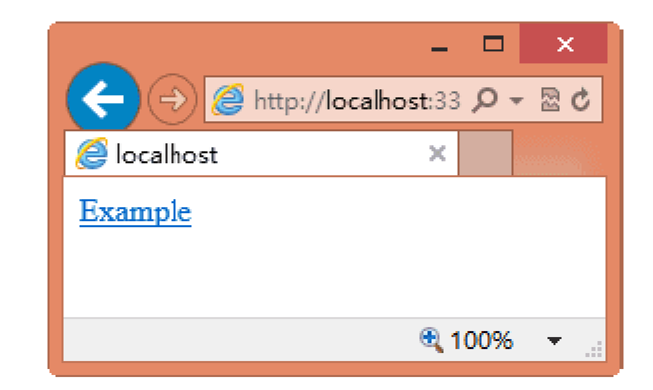


图 3-3　example3-2-1.aspx 运行效果

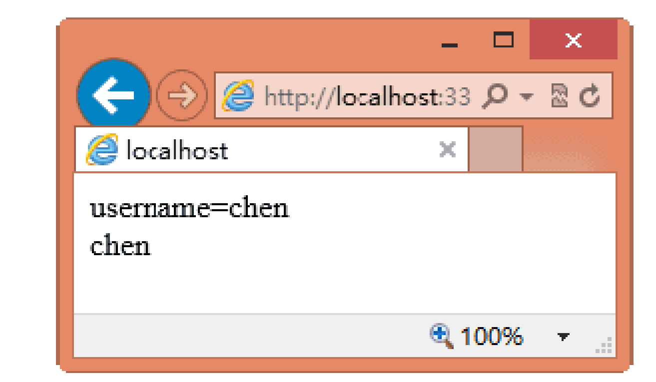


图 3-4　example3-2-2.aspx 运行效果

3.3.3　Application 对象

人们最常用的保存变量的内容的方法是使用文件，但是毕竟对文件的操作是比较麻烦的事情，有没有更简单的方法呢？其中一种比较简单的方法就是使用 Application 对象来保存希望传递的变量。由于在整个应用程序生存周期中，Application 对象都是有效的，所以在不同的页面中都可以对它进行存取，就像使用全局变量一样方便。

在 ASP.NET 环境中，Application 对象来自 HttpApplicationState 类，它可以在多个请求、连接之间共享公用信息，也可以在各个请求连接之间充当信息传递的管道，Application 对象常用的属性和方法如表 3-6 和表 3-7 所示。

表 3-6　Application 属性列表

属　　性	方　　法
AllKeys	获得访问 HttpApplicationState 集合的所有键
Contents	获得 HttpApplicationState 对象的引用
Count	获得 HttpApplicationState 集合的数量
Item	通过名称和索引访问 HttpApplicationState 集合
Keys	获得访问 HttpApplicationState 集合的所有键，从 NameObjectCollectionBase 继承
StaticObjects	获得所有使用<object>标签声明的应用程序集对象

表 3-7　Application 方法列表

方　　法	说　　明
Add	添加一个新的对象到 HttpApplicationState 集合
Clear	清除 HttpApplicationState 集合中的所有对象
Get	通过索引和名字获得 HttpApplicationState 对象
GetKey	通过索引获得一个 HttpApplicationState 名称
Lock	锁定访问 HttpApplicationState 变量
UnLock	取消锁定，一般情况下需要操作 Application 变量则设置为 Lock，操作完成后则设置为 Unlock
Remove	从 HttpApplicationState 集合删除一个对象
RemoveAll	删除 HttpApplicationState 集合所有对象
RemoveAt	根据索引删除一个 HttpApplicationState 对象
Set	更新一个 HttpApplicationState 变量

下面对该对象常用的一些属性和方法进行说明。

- Add 方法，加入一个对象到 Application 对象的 Stat 集合中。如下所示。

```
Application.Add("string1","test")
```

表示向 Application 的 Stat 集合中加入一个名为 string1 的值为 test 的字符串，其实它的效果和 Application("string1")="test"以及 Application.Item("sting1")="test"是一样的。

- Remove 方法，根据给出的标识从 Application 对象的 Stat 集合中删去，如下所示。

```
Application.Remove("sting1")
```

表示把标识为 sting1 的共享对象 sting1 从 Application 对象的 Stat 集合中删去。使用它可以清除用 Add 方法添加的对象。

- RemoveAll 方法，把 Application 对象 Stat 集合中的所有对象清除。在对属性的使用举例中，已经见过了它的用法，但是需要小心，不提倡使用它，因为在编程中并不清楚是否有其他页面要依赖于某个 Application 的公用变量，一旦清除将造成不可预知的错误。
- Clear 方法，作用和 RemoveAll 方法一样。
- Get 方法，允许使用名字标识或者是下标来取得 Application 对象 Stat 集合中的对象元素。如下所示。

```
Tmp=Application.Get(0)
```

- Set 方法，修改 Application 对象 Stat 集合中指定标识所对应的对象值。如下所示。

```
Application("sting1","try")
```

表示把开始为 sting1 变量设置的值 test 改为 try。

【例 3-3】在线统计网站访问人数，演示 Application 对象一些方法的使用。

(1) 在 WebSite3 网站中添加一个名为 example3-3.aspx 的网页。

(2) 在 example3-3.aspx 页面的<body>中添加以下代码。

```
<form id="form1" runat="server">
<div>
    <p>当前网站访问人数是：
    <asp:Label ID="labContent" runat="server" Text="Label"></asp:Label>
        </p>
</div>
</form>
```

(3) 在 example3-3.aspx.cs 页面中添加下面代码。

```
protected void Page_Load(object sender, EventArgs e)
{
    Application.Lock();
    Application["usercount"] = (Convert.ToInt32(Application["usercount"]) + 1).ToString();
    Application.UnLock();
    labContent.Text = Application["usercount"].ToString();
}
```

(4) 程序运行效果如图 3-5 所示。

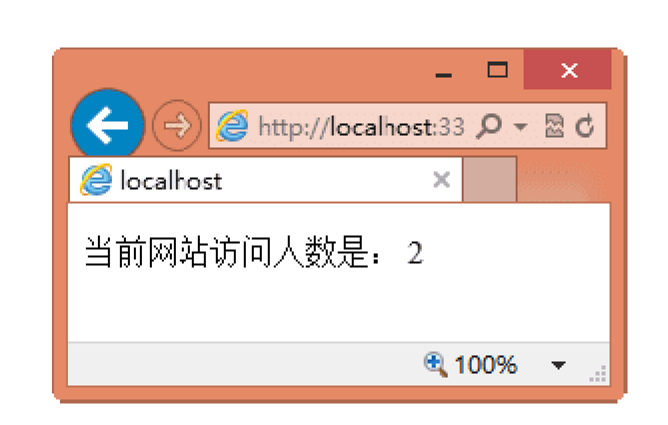


图 3-5　example3-3.aspx 运行效果

3.3.4　Session 对象

Session 对象的作用也是用于储存特定的信息，但是它和 Application 对象在储存信息所使用的对象是完全不同的。Application 对象储存的是共享信息，而 Session

储存的信息是局部的，是随用户不同而不同的。如果只需要在不同页中共享数据，而不是需要在不同的客户端之间共享数据就可以使用 Session 对象。

Session 的生命周期是有限的(默认值为 20 分钟)，它可以使用 Timeout 属性进行设置。在 Session 的生命周期内，Session 的值是有效的。如果用户在大于生命周期的时间里没有再访问应用程序，Session 就会自动过期，Session 对象将会被 CLR 释放，其储存的数据信息将永远不再存在。Session 对象的常用属性和方法如表 3-8 和表 3-9 所示。

表 3-8　Session 对象常用属性列表

方　法	说　明
CodePage	获得或设置字符集标识
Contents	获得当前 Session 状态对象的引用
CookieMode	获得当前的 Cookie 模式，以确定系统是否要将 Session 配置为不需要 Cookie 支持
Count	Session 状态集合的总数
IsCookieless	是否需要 Cookie 支持，如果需要就可以将 Session ID 保存在 Cookie 中，如果不需要就必须嵌入在 URL 中
IsNewSession	标志当前 Session 是否是新的 Session
IsReadOnly	是否只读
IsSynchronized	是否同步
Item	通过索引获得或者设置单个 Session 值
Keys	获得 Session 集合的所有键
LCID	获得和设置当前 Session 的本地标识符
Mode	获得当前的 Session 模式
SessionID	获得 Session 的唯一编号，为了区别不同的会话，系统会为每一个会话分配一个唯一的 ID
StaticObjects	获得在 Global.asax 中以<object Runat="Server" Scope="Session" />声明的对象集合
Timeout	获得和设置会话超时时间，如果客户端在连续一个时间段内没有反应，就自动清除会话，断开连接，Timeout 就是这个时间段

表 3-9　Session 对象常用方法列表

方　法	说　明
Add	添加一个新对象到 HttpApplicationState 集合
Clear	清除 HttpApplicationState 集合中的所有对象
Get	通过索引和名字获得 HttpApplicationState 对象
Abandon	清除当前会话
Copyto	复制 Session 状态集合到一个一维数组
Remove	从 HttpApplicationState 集合删除一个对象
RemoveAll	删除 HttpApplicationState 集合所有对象
RemoveAt	根据索引删除一个 HttpApplicationState 对象

【例 3-4】 演示 Session 方法的使用。

(1) 在 WebSite4 网站中添加名为 example3-4-1.aspx 和 example3-4-3.aspx 的两个网页。

(2) 在 example3-4-1.aspx 的<body>中添加如下代码。

```
<form id="form1" runat="server">
    <div>
    <p>用户名:
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <asp:Button ID="Button1" runat="server" Text="提  交" />
    </p>
    </div>
    </form>
```

(3) 在 example3-4-2.aspx 的<body>中添加如下代码。

```
<form id="form1" runat="server">
    <div>
        <p>用户名:
        <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
            </p>
    </div>
    </form>
```

(4) 在 example3-4-1.aspx.cs 页面中添加如下代码。

```
protected void Page_Load(object sender, EventArgs e)
    {
        if (TextBox1.Text!="")
        {
        Session["username"] = TextBox1.Text;
        Response.Redirect("example3-4-2.aspx");
        }
    }
```

(5) 在 example3-4-2.aspx.cs 页面的中添加如下代码。

```
protected void Page_Load(object sender, EventArgs e)
    {
        if (Session["username"] != null)
        {
            Label1.Text = Session["username"].ToString();
        }
        else
        {
            //请输入用户名
            Response.Redirect("example3-4-1.aspx");
        }
    }
```

(6) 程序运行效果如图 3-6 和图 3-7 所示，从图中可以看到利用 Session 对象可以存储用户的相关信息，并在需要时进行读取显示。

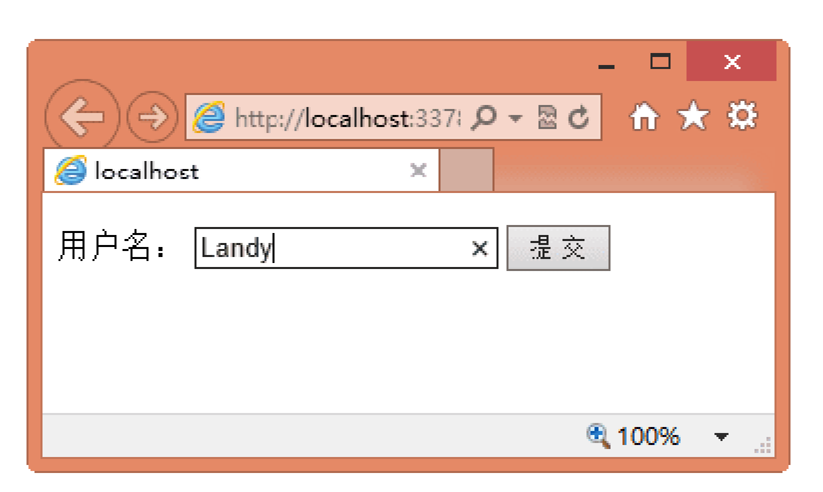


图 3-6　example3-4-1.aspx 运行效果

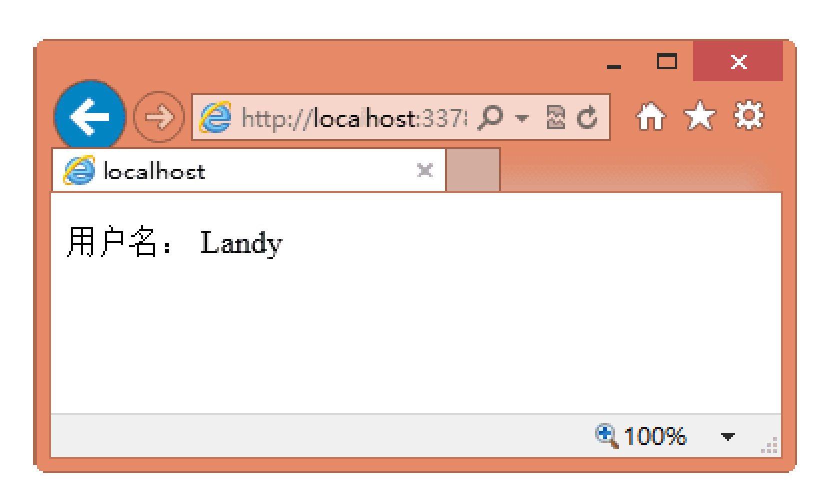


图 3-7　example3-4-2.aspx 运行效果

3.3.5　Server 对象

Server 对象即服务器对象，就是在服务器上工作的一个对象，用于建立 COM 对象实例、处理应用程序错误，在页面之间传递控件，获取最新出错信息，对 HTML 进行编码和解码等。Server 对象提供许多访问的方法和属性帮助程序有序地执行，主要属性和方法如表 3-10 和表 3-11 所示。

表 3-10　Server 对象常用属性

名　称	说　明
MachineName	获得服务器计算机名称
ScriptTimeout	获得和设置请求超时的事件

表 3-11　Server 对象常用方法

方　法	说　明
CreateObject	建立一个 COM 组件对象的实例
Execute	执行指定资源并返回
Transfer	结束当前页执行，转到其他页执行
HtmlEncode	进行 HTML 编码
HtmlDecode	进行 HTML 解码
MapPath	对虚拟目录进行物理映射
UrlEncode	进行 URL 编码
UrlDecode	进行 URL 解码

Server 对象的一个重要功能是对字符进行 URL 和 HTML 的编码和解码。URL 编码的目的是保证所有浏览器能够正确地传输 URL 路径，一些特殊字符如？、&、/、空格和中文字符等，在传输时都有可能让浏览器发生错误。先通过编码再将其传输，在需要使用时又通过解码将其还原。HTML 编码的作用是将所有字符全部转化为 HTML 中能够用来显示的字符。例如，<p>这样的字符如果直接显示就是一个段落，如果转换以后就会变成<p>浏览时就可以正确显示出<p>，就不会造成一些错误。下面对一些常用的方法进行解释。

1. HtmlEncode、HtmlDecode 方法

HtmlEncode 方法的作用是将代码中的 HTML 标签进行了转码，目的是显示这段源代

码，而不是去执行它。

例如，当用户需要显示一个<HTML>时，如果写 Response.Write("<html>");的话，是不能达到目的的，需要如下这样来写才能达到目的。

```
Response.Write Server.HtmlEncode("<HTML>");
```

这时，查看源文件，就可以看到，使用该方法已经把<HTML>字符串转换成为<html>。但是，在显示时还是显示<HTML>，这样，使用 HtmlEncode 方法就可以让用户随心所欲地显示用户所要显示的内容，而不会和用户的页面混合在一起。

HtmlDecode 方法的作用与 HtmlEncode 方法的作用正好相反，它会将被 HTML 编码的代码解码，恢复代码本来面目。

2. UrlEncode、UrlDeconde 方法

UrlEncode 方法与 HtmlEncode 方法类似，将指定的代码以 URL 格式进行编码，把 URL 中的所有特殊字符转化成为非功能的等价内容。通常在把 URL 作为查询字符串时使用。用下面的方法可以看出这些特定的字符被转化成的内容。

```
Response.Write(Server.UrlEncode(http://www.abc.com));
```

执行完成后，显示如下。

```
http%3A%2F%2Fwww%2Eabc%2Ecom
```

UrlDecode 和 UrlEncode 刚好相反，该方法将 URL 格式代码进行解码。

3. MapPath 方法

Server.MapPath 方法将虚拟路径转换为绝对路径。这种方法在需要包含或执行其他的文件并需要制定路径名，但路径名又常常发生变化的情况下使用。如下所示。

```
StingsPath=Server.MapPath("/");
```

4. Transfer(path)方法

终止当前程序的执行，进入 path 所指的程序。该方法可以把控制传递出去，可以把原来页面的所有内置对象和这些对象的状态都传递给新的页面，如 Request 对象的查询字符串。使用这种方法还可以把一个大的程序划分成小的模块，然后用 Transfer 方法把各个模块联系起来。

5. Execute(path)方法

在当前程序中执行 path 所指定的程序，它与 Transfer 方法的不同之处在于，当 path 所指的程序运行完毕后，将继续执行当前程序中后面的代码。

3.3.6　ViewState(视图状态)对象

ViewState 对象是状态管理中常用的一种对象，可以用来保存页和控件的值，下面对其进行介绍。

1. ViewState 简介

视图状态是 ASP.NET 页框架默认情况下用于保存往返过程之间的页面信息以及控件值的方法。当呈现页的 HTML 形式时，需要在回发过程中保留的页的当前状态和值将被序列化为 Base64 编码的字符串，并输出到视图状态的隐藏字段中。通过实现自定义的 PageStatePersiste 类以存储页数据，也可以更改默认行为并将视图状态存储到另一个位置，如 SQL Server 数据库。

程序员可以通过使用页面的 ViewState 属性将往返过程中的数据保存到 Web 服务器端，然后利用自己的代码访问视图状态。ViewState 属性是一个包含密钥/值对(其中包含视图状态数据)的字典。

视图状态中存储的常见数据类型有：字符串、整数、布尔值、Array 对象、ArrayList 对象、哈希表和泛型对象等。

2. 使用 ViewState 时的注意事项

使用 ViewState 可以带来很多方便，但是也有一些问题是需要注意的。

- 视图状态提供了特定 ASP.NET 页面的状态信息。如果需要在多个页上使用信息，或者需要在访问网站时保留信息，则应当使用另一个方法(如应用程序状态、会话状态或个性化设置)来维护状态。
- 视图状态信息将序列化为 XML，然后使用 Base64 编码进行编码，这将生成大量的数据。将页回发到服务器时，视图状态信息将作为页回发信息的一部分发送。如果视图状态包含大量信息，则会影响页的性能。因此，建议使用一些典型数据来测试页性能，确定视图状态的大小是否是导致应用程序性能问题的“瓶颈”。
- 虽然使用视图状态可以保存页和控件的值，但是在某些情况下，需要关闭视图状态。例如，使用 GridView 控件显示数据，单击 GridView 控件的下一页按钮。此时，GridView 控件呈现的数据已经不再是前一页的数据，那么如果使用视图状态将前一页数据保存下来，不仅没有必要而且还会生成大量隐藏字段，增大页面的体积，此时应当关闭视图状态以移除由 GridView 控件生成的大量隐藏字段。

另一个重要的考虑因素是，如果隐藏字段中的数据量过大，某些代理的防火墙将禁止访问包含这些数据的页。由于所允许的最大数据量随所采用的防火墙和代理的不同而不同，因此大量隐藏字段可能会导致偶发性问题。为了帮助避免这一问题，如果 ViewState 属性中存储的数据量超过了页的 MaxPageStateFieldLength 属性中指定的值，该页会将视图状态拆分为多个隐藏字段，可以使每个单独字段的大小在防火墙拒绝的大小以下。

【例 3-5】演示如何使用 ViewState 对象，从视图状态中读取值和将输入的新值保存在视图状态中，并可以进行读取显示。

(1) 在 WebSite3 网站中添加一个名为 example3-5.aspx 的网页。

(2) 在 example3-5.aspx 中添加以下代码。

```
<form id="form1" runat="server">
    <div>
```

```
            <asp:TextBox ID="TextBox1" runat="server" ViewStateMode="Enabled"></asp:TextBox>
            <br />
            <asp:Button ID="Button1" runat="server"  Text="保存 ViewState 中的值"
ViewStateMode="Enabled" OnClick="Button1_Click" />
            <br />
            <asp:Label ID="Label1" runat="server" Text="Label"
ViewStateMode="Enabled"></asp:Label>
            <br />
            <asp:Button ID="Button2" runat="server" Text="读取 ViewState 中的值"
ViewStateMode="Enabled" OnClick="Button2_Click" />
        </div>
        </form>
```

(3) 在 example3-5.aspx.cs 页面中添加下面代码。

```
public partial class Default2 : System.Web.UI.Page
{
    string VSString = "basketball";
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            ViewState.Add("favorite", VSString);
        }
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        VSString = this.TextBox1.Text;
        ViewState["favorite"] = VSString;
    }
    protected void Button2_Click(object sender, EventArgs e)
    {
        if (ViewState["favorite"] != null)
        {
            this.Label1.Text = ViewState["favorite"].ToString();
        }
        else
        {
            this.Label1.Text = "查看的 ViewState 值不存在";
        }
    }
}
```

(4) 程序运行效果如图 3-8 和图 3-9 所示。

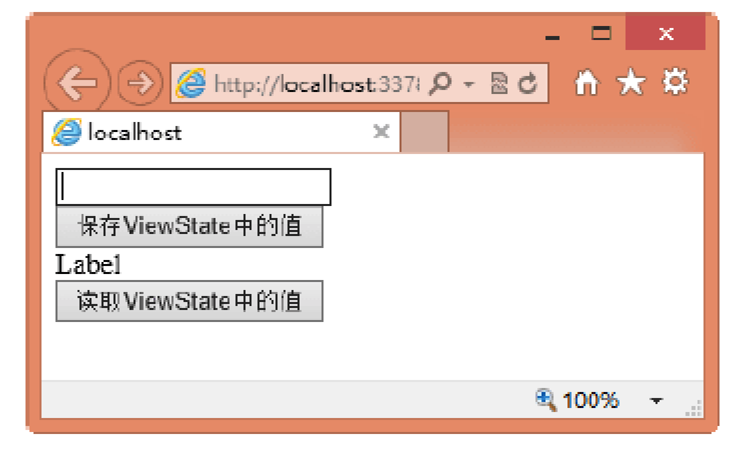


图 3-8　读取 ViewState 中的初值

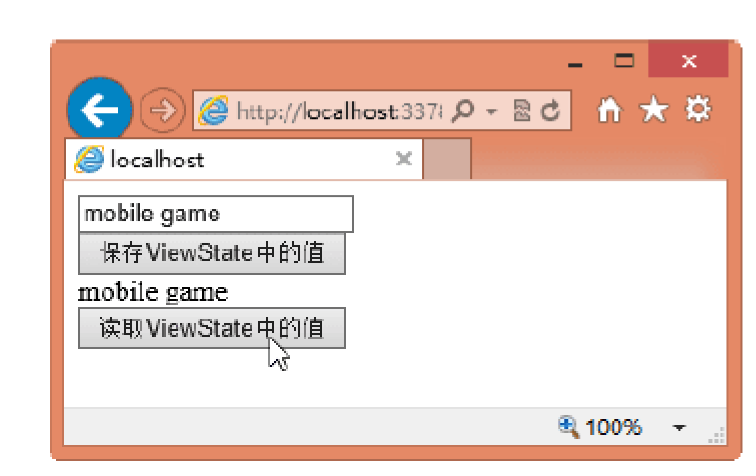


图 3-9　读取 ViewState 中新保存的值

3.3.7　Cookie 对象

Cookie 对象是比较常用的一种对象，可以用来保存一些常用的信息。

1. Cookie 对象简介

Cookie 俗称“小甜饼”，通常是用于存储少量的浏览者的信息，如浏览者的喜好、用户名、Email 地址等信息，以便于当浏览者再次登录网站时，不必再次填写这些信息。Cookie 其实只是一些小文本，将一些用户信息储存在客户端的机器中，它全部存储于 Windows 目录下的 Cookie 文件夹中，以便于在每次请求时被服务器在设定的时期内进行读取。Cookie 的储存大小是有限制的，一般浏览器会将其大小控制在 4096 个字节以内。

Cookie 与网站关联，而不是与特定的页面关联。因此，无论用户请求站点中的哪一个页面，浏览器和服务器都将交换 Cookie 信息。用户访问不同站点时，各个站点都可能会向用户的浏览器发送一个 Cookie，浏览器会分别存储所有 Cookie。

每次用户访问站点时，Web 应用程序都可以读取 Cookie 信息。当用户请求站点中的页面时，应用程序发送给该用户的不仅仅是一个页面，还有一个包含日期和时间的 Cookie，用户的浏览器在获得页面的同时也获得了该 Cookie，并将它存储在用户硬盘上的文件夹中。以后，如果该用户再次请求站点中的页面，当该用户输入 URL 时，浏览器便会在本地硬盘上查找与该 URL 关联的 Cookie。例如，当用户登录某些网站的邮箱后，如果在 Cookie 中记录了用户名这个信息，那么在 Cookie 信息失效以前，该用户在同一台计算机再次登录时就不需要提供用户名。

2. Cookie 的限制

大多数浏览器支持最大为 4096 字节的 Cookie。由于这限制了 Cookie 的大小，最好用 Cookie 来存储少量数据，或者存储用户 ID 之类的标识符。用户 ID 随后便可用于标识用户，以及从数据库或其他数据源中读取用户信息。

浏览器还限制站点可以在用户计算机上存储的 Cookie 的数量。大多数浏览器只允许每个站点存储 20 个 Cookie，如果试图存储更多的 Cookie，则存放最早的 Cookie 便会被覆盖掉。有些浏览器还会对它们将接受的来自所有站点的 Cookie 总数做出绝对限制，通常为 300 个。另外一个可能遇到的 Cookie 限制是用户可以将其浏览器设置为拒绝接受 Cookie。

3. 读取 Cookie

浏览器向服务器发出请求时，会随请求一起发送该服务器的 Cookie。在 ASP.NET 应用程序中，可以使用 HttpRequest 对象读取 Cookie，该对象可用作 Page 类的 Request 属性使用。HttpRequest 对象的结构与 HttpResponse 对象的结构基本相同，因此，可以从 HttpRequest 对象中读取 Cookie，并且读取方式与将 Cookie 写入 HttpResponse 对象的方式基本相同。

4. 修改和删除 Cookie

服务器不能直接修改 Cookie。如果要修改 Cookie，必须首先创建一个具有新值的新 Cookie。如下所示。

```
HttpCookie cookie=new HttpCookie("hello")
```

建立起一个名为 hello 的 HttpCookie 实例。

建立实例后，将给其赋值。在一个 Cookie 中可以储存一个值，也可以储存多个值。通过设置 Cookie 的 Value 属性值，可以在 Cookie 中储存一个值。代码如下。

```
Cookie.Value="test"
```

通过 Cookie 的 Values 集合，可以在同一个 Cookie 中储存多个值。代码如下。

```
HttpCookie cookie=new HttpCookie("test");
Cookie.Values.Add("Admin", "刘一");
Cookie.Values.Add("Member1", "李四");
Cookie.Values.Add("Member2", "王五");
Cookie.Values.Add("Member3", "张三");
```

Values 集合使用的 Add 方法中第一个参数为关键字(Key)，第二个参数是设置的值(Value)。

删除 Cookie 是修改 Cookie 的一种形式。由于 Cookie 在用户计算机中，因此无法将其直接移除。但是，可以让浏览器来删除 Cookie，该技术是创建一个与要删除的 Cookie 同名的新 Cookie，并将该 Cookie 的到期日期设置为过去的某个日期，当浏览器检查 Cookie 的到期日期时，浏览器便会丢弃这个现已过期的 Cookie。

5. 确定浏览器是否接受 Cookie

用户可将其浏览器设置为拒绝接受 Cookie。虽然不能向客户端写入 Cookie 信息，但是不会引发任何错误。同样，浏览器也不向服务器发送有关其当前 Cookie 设置的任何信息。确定 Cookie 是否被接受的一种方法时尝试编写一个 Cookie，然后再尝试读取该 Cookie。如果无法读取已编写的 Cookie，则可以假定浏览器不接受 Cookie。

【例 3-6】通过一个例子来说明如何编写 Cookie 与读取 Cookie。

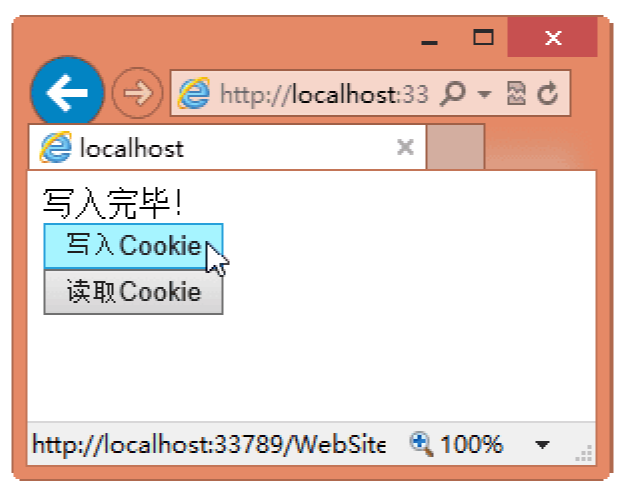
(1) 在 WebSite3 网站中添加一个名为 example3-6.aspx 的网页，其<body>标记中的内容如下。

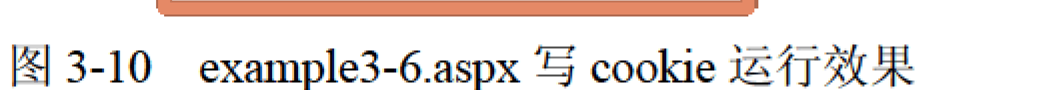
(2) 在 example3-6.aspx.cs 页面中添加如下代码。

```
protected void Button1_Click(object sender, EventArgs e)
{
    Response.Cookies["userName"].Value = "BUAA University";
    Response.Cookies["userName"].Expires = DateTime.Now.AddDays(1);
    this.Label1.Text = "写入完毕！ ";
}
protected void Button2_Click(object sender, EventArgs e)
```

```
    {
        if (Request.Cookies["userName"] != null)
        {
            Label1.Text = Server.HtmlEncode(Request.Cookies["userName"].Value);
        }
    }
```

(3) 程序运行效果如图 3-10 和图 3-11 所示。




图 3-10　example3-6.aspx 写 cookie 运行效果

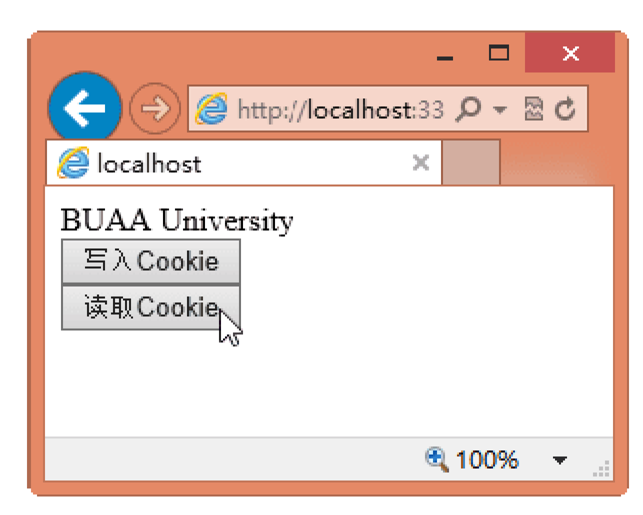


图 3-11　example3-6.aspx 读 cookie 运行效果

3.4　ASP.NET 配置管理

使用 ASP.NET 配置系统的功能，可以配置整个服务器上的所有 ASP.NET 应用程序、单个 ASP.NET 应用程序和各个页面或应用程序子目录，也可以配置各种具体的功能，如身份验证模式、页缓存、编译器选项、自定义错误、调试和跟踪选项等。

3.4.1　web.config 文件介绍

ASP.NET 提供了一个丰富而可行的配置系统，以帮助管理人员轻松快速地建立自己的 Web 应用环境。

Web 配置文件 web.config 是 Web 应用程序的数据设定文件，它是一份 XML 文件，内含 Web 应用程序相关设定的 XML 标记，可以用来简化 ASP.NET 应用程序的相关设定。该文件位于 Web 应用程序的任何目录中，统一命名为 web.config，它决定了所在目录及其子目录的配置信息，并且子目录下的配置信息覆盖其父目录的配置，即子目录如果没有 web.config 文件，就是继承父目录 web.config 文件的相关设定；如果子目录有 web.config 文件，就会覆盖父目录 web.config 文件的相关设定。在运行状态下，ASP.NET 会根据远程 URL 请求，把访问路径下的各个 web.config 配置文件叠加，产生一个唯一的配置集合。

举例来说，一个对 URL:http://localhost\website\ownconfig\test.aspx 的访问，ASP.NET 会根据以下顺序来决定最终的配置情况：

(1) .\Microsoft.NET\Framework\{version}\web.config (默认配置文件)

(2) .\webapp\web.config(应用的配置)

(3) .\webapp\ownconfig\web.config(自己的配置)

web.config 是 aspx 区别于 asp 的一个方面，用户可以用这个文件配置自己的很多信息。

ASP.NET 允许配置内容可以和静态内容、动态页面和商业对象放置在同一应用的目录结构下。当管理人员需要安装新的 ASP.NET 应用时，只需要将应用目录复制到新的机器上即可。

ASP.NET 的配置系统具有以下优点。

- ASP.NET 的配置内容以纯文本方式保存，可以以任意标准的文本编辑器、XML 解析器和脚本语言解释、修改配置内容。
- ASP.NET 提供了扩展配置内容的架构，以支持第三方开发者配置自己的内容。
- ASP.NET 配置文件的更改被系统自动监控，无须管理人员手工干预。

3.4.2　配置文件的语法规则

1. 标识

配置内容被置于 web.config 文件中的标记<configuration>和</configuration>之间。格式如下。

```
<configuration>
配置内容
…
</configuration>
```

2. 配置段

具体定义配置的内容，以供应用使用。Web 配置文件是一份 XML 文件，在 XML 标记的属性就是设定值，标记名称和属性值格式是字符串，第一个开头字母是小写，之后每一字头是大写，例如<appSetting>。Web 配置文件的范例如下所示。

以下例子定义了一个 httpmodules 配置段，设置了系统 http 相关的处理模块。

```
<configuration>
      <appSettings>
         <add key="dbType" value="Access Database"/>
      </appsettings>
      <connectionsStrings>
<add name="provider"
connectionString="Microsoft.Jet.OLEDB.4.0;"/>
<add name="database"
connectionString="/chart7/Exam.mdb"/>
</connectionsStrings>
        <system.web>
<sessionState cookieless="false" timeout="10"/>
<compilation defaultLanguage="C#" debug="true"/>
<globalization
                 fileEncoding="gb2312"
 requestEncoding="gb2312"
culture= "zh-CN"/>
                 <customErrors mode="RemoteOnly"/>
           </system.web>
</configuration>
```

可以看到，在这段配置信息当中这是一个基于 XML 格式的文件，根标记是<configuration>，所有的配置信息均被包括在<configuration>及</configuration>标签中间，其子标记<appSettings>、<connectionsStrings>和<system.web>是各设定区段。在<system.web>下的设定区段属于 ASP.NET 相关设定。

在 Web 配置文件的<appSettings>区段可以创建 ASP.NET 程序所需要的参数，每个<add>标记可以创建一个参数，属性 key 是参数名称，value 是参数值。ASP.NET 2.0 以后新增<connectionStrings>区段，可以指定数据库连接字符串，在<connectionStrings>标记的<add>子标记也可以创建连接字符串，属性 name 是名称，connectionStrings 是连接字符串的内容，表 3-12 列出了常用设定区段标记的说明。

表 3-12　常用设定区段标记说明

设 定 区 段	说　明
<AnonymousIdentification>	控制 Web 应用程序的匿名用户
<Authentication>	设定 ASP.NET 的验证方式
<Authorization>	设定 ASP.NET 用户授权
<BrowserCaps>	设定浏览程序兼容组件 HttpBrowserCapabilities
<Compilation>	设定 ASP.NET 应用程序的编译方式
<CustomErrors>	设定 ASP.NET 应用程序的自动错误处理
<Globalizations>	关于 ASP.NET 应用程序的全球化设定，也就是本地化设定
<HttpHandlers>	设定 HTTP 处理是对应到 URL 请求的 HttpHandler 类
<HttpModules>	创建\删除或清除 ASP.NET 应用程序的 HTTP 模块
<HttpRuntime>	设定 ASP.NET HTTP 运行时的配置，这些配置决定如何处理对 ASP.NET 应用程序的请求
<MachineKey>	设定在使用窗体基础验证的 Cookie 数据时，用来加码和解码的金钥匙
<Membership>	设定 ASP.NET 的 Membership 机制
<Pages>	设定 ASP.NET 程序的相关设定，即 Page 指引命令的属性
<Profile>	设定个人化信息的 Profile 对象
<Roles>	设定 ASP.NET 的角色管理
<SessionState>	设定 ASP.NET 应用程序的 Session 状态 HttpModule
<SiteMap>	设定 ASP.NET 网站导航系统
<WebParts>	设定 ASP.NET 应用程序的网页组件
<WebServices>	设定 ASP.NET 的 Web 服务

3.4.3　Global.asax 文件介绍

作为网络应用程序，程序在执行之前有时需要初始化一些重要的变量，而且这些工作必须发生在所有程序执行之前，ASP.NET 的 Global.asax 文件便是为此目的而设计的。每个 ASP.NET 应用程序都可以有一个 Global.asax 文件。一旦将其放在适当的虚拟目录中

ASP.NET 就会把它识别出来并且会自动使用该文件。

另外，由于 Global.asax 在网络应用程序中的特殊地位，它被存放的位置也是固定的。必须存放在当前应用所在的虚拟目录的根目录下。如果放在虚拟目录的子目录中，Global.asax 文件将不会起任何作用。

ASP.NET 在应用程序中添加了“全局应用程序类”，也就是 Global.asax。该文件是应用程序用来保持应用程序级的事件、对象和变量的。一个 ASP.NET 应用程序只能有一个 Global.asax 文件，位于应用程序根目录下。

默认添加的 Global.aspx 文件如下所示。

```
<%@ Application Language="C#"%>
<script runat="server">
    void Application_Start(object sender,EventArgs e)
    {
        //在应用程序启动时运行的代码
    }
        void Application_End(object sender, EventArgs e)
    {
        //在应用程序关闭时运行的代码
    }
     void Application_Error(object sender, EventArgs e)
    {
        //在出现未处理的错误时运行的代码
    }
    void Session_Start(object sender, EventArgs e)
    {
        //在新会话启动时运行的代码
    }
</script>
```

在窗体页中，只能处理单个页面的事件，而在 Global.asax 文件中可以处理整个应用程序中的事件。除了上述代码模板中列举的事件以外，在 Global.asax 文件中还可以加入其他事件的处理函数。如表 3-13 所示的是可以在 Global.asax 中处理的事件。

表 3-13　Global.asax 中处理的事件

事　　件	说　　明
Application_Start	在应用程序接收到第一个请求时调用，通常在此函数中定义应用程序级变量或状态
Session_Start	类似与 Application_Start，不过是针对每个客户端第一次访问应用程序时调用
Application_BeginRequest	虽然在 VWD 的代码模板中没有该事件的处理，不过可以在 Global.asax 中添加。该事件是在每个请求到达服务器，并且在处理该请求前，会触发的事件
Application_AuthenticateRequest	每个请求都会触发该事件，并且可以在此函数中设置自定义的验证
Application_Error	在应用程序中抛出任何错误时都会触发该事件。通常在此函数中提供应用程序级的错误处理或者记录错误事件

(续表)

事　件	说　明
Session_End	以进程内模式使用会话状态时，如果用户离开应用程序将会触发该事件
Application_End	应用程序关闭时触发该事件。该函数很少使用，因为 ASP.NET 可以很好地关闭和清除内存对象

与页面指令一样，Global.asax 文件也可以使用应用程序指令，这些指令都可以包含特定于该指令的一个或多个属性/值对。下面列出了 ASP.NET 中支持的应用程序指令。

- @Application：定义 ASP.NET 应用程序编译器所使用的应用程序特定的属性。该指令只能在 Global.asax 文件中使用。
- @Import：显式将命名空间导入到应用程序中。
- @Assembly：在分析时将程序集链接到应用程序。

3.5　本章小结

本章内容是认识了解使用 ASP.NET 的一个关键点。在用 ASP.NET 进行程序开发时，其代码组织模式和页面的对象模型往往会贯穿整个开发过程，了解其基本运作方式有利于项目的顺利开发。

本章主要介绍 ASP.NET 网站技术基础，内容包括 ASP.NET 文件类型，文件夹等；页面管理基本技术，包括页面代码模式，页面的往返与处理，页面的生命周期以及相关的事件、方法，页面指令等；ASP.NET 的常用内置函数介绍，包括 Response、Request、Application、Session、Server、ViewState、Cooke；ASP.NET 配置管理，包括 Web.config 和 Global.asax 的配置。学习完本章内容后，读者应能掌握 ASP.NET 网站目录结构及作用。

3.6　思考和练习

1. ASP.NET 网站目录中，有哪些特殊的目录？
2. 常用的页面代码模式有几种？请简单叙述。
3. 请谈谈页面往返和处理的机制。
4. ASP.NET 的常用内置对象有哪些？功能如何？
5. 请开发者谈谈对 Web.config 和 Global.asax 的理解。

第4章　C#语法基础

C#是.NET 平台下最重要的语言，也是微软技术体系中最受欢迎的语言。微软在.NET 中为 C#提供了大量的新特性，以帮助开发人员快速、准确、灵活地实现各种业务功能，并满足多种环境下的开发需求。本章将讲解 C#中最基本的语法知识，这些语法知识可以为开发人员打下坚实的开发基础，以便更好地使用 C#语言去开发基于 C#语言的 Web 应用程序。为了讲解 C#语法知识，本章将使用 Microsoft Visual Studio 2012 作为环境进行介绍。

本章的学习目标：

- 掌握变量、常量和表达式的使用。
- 掌握算术、关系、逻辑、赋值运算符的使用及优先级对运算顺序的影响。
- 掌握 if、switch、for、foreach、while 流程控制语句的使用。
- 掌握方法的定义、参数、返回值、重载等操作。
- 掌握面向对象编程中的类、对象、接口的使用以及继承、多态的概念。

4.1　C#语言特点

Microsoft.NET(以下简称.NET)框架是微软提出的新一代 Web 软件开发模型，C#语言是.NET 框架中新一代的开发工具。C#语言是一种现代、面向对象的语言，它简化了 C++语言在类、命名空间、方法重载和异常处理等方面的操作，它摒弃了 C++的复杂性，更易使用，更少出错。它使用组件编程，和 VB 一样容易使用。C#语法和 C++和 JAVA 语法非常相似，如果读者用过 C++和 JAVA，学习 C#语言应是比较轻松的。

用 C#语言编写的源程序，必须用 C#语言编译器将 C#源程序编译为中间语言(MicroSoft Intermediate Language，简称 MSIL)代码，形成扩展名为 exe 或 dll 文件。中间语言代码不是 CPU 可执行的机器码，在程序运行时，必须由通用语言运行环境(Common Language Runtime，简称 CLR)中的即时编译器(JUST IN Time，简称 JIT)将中间语言代码翻译为 CPU 可执行的机器码，由 CPU 执行。CLR 为 C#语言中间语言代码运行提供了一种运行时环境，C#语言的 CLR 和 JAVA 语言的虚拟机类似。这种执行方法使运行速度变慢，但带来其他一些好处，主要如下。

- 通用语言规范(Common Language Specification，简称 CLS)：.NET 系统包括 C#、C++、VB、J#语言，他们都遵守通用语言规范。任何遵守通用语言规范的语言源程序，都可编译为相同的中间语言代码，由 CLR 负责执行。只要为其他操作系统编制相应的 CLR，中间语言代码也可在其他系统中运行。

- 自动内存管理：CLR 内建垃圾收集器，当变量实例的生命周期结束时，垃圾收集器负责收回不被使用的实例占用的内存空间。不必像 C 和 C++语言，用语句在堆中建立的实例，必须用语句释放实例占用的内存空间。也就是说，CLR 具有自动内存管理功能。
- 交叉语言处理：由于任何遵守通用语言规范的语言源程序，都可编译为相同的中间语言代码，不同语言设计的组件，可以互相通用，可以从其他语言定义的类派生出本语言的新类。由于中间语言代码由 CLR 负责执行，因此异常处理方法是一致的，这在调试一种语言调用另一种语言的子程序时，显得特别方便。
- 增加安全：C#语言不支持指针，一切对内存的访问都必须通过对象的引用变量来实现，只允许访问内存中允许访问的部分，这就防止病毒程序使用非法指针访问私有成员。也避免指针的误操作产生的错误。CLR 执行中间语言代码前，要对中间语言代码的安全性，完整性进行验证，防止病毒对中间语言代码的修改。
- 版本支持：系统中的组件或动态链接库可能要升级，由于这些组件或动态联接库都要在注册表中注册，由此可能带来一系列问题。例如，安装新程序时自动安装新组件替换旧组件，有可能使某些必须使用旧组件才可以运行的程序，使用新组件运行不了。在.NET 中这些组件或动态联接库不必在注册表中注册，每个程序都可以使用自带的组件或动态联接库，只要把这些组件或动态联接库放到运行程序所在文件夹的子文件夹 bin 中，运行程序就自动使用在 bin 文件夹中的组件或动态联接库。由于不需要在注册表中注册，软件的安装也变得容易了，一般将运行程序及库文件复制到指定文件夹中就可以了。
- 完全面向对象：不像 C++语言，即支持面向过程程序设计，又支持面向对象程序设计，C#语言是完全面向对象的，在 C#中不再存在全局函数、全局变量，所有的函数、变量和常量都必须定义在类中，避免了命名冲突。C#语言不支持多重继承。

4.2　建立 C#应用程序

为了方便讲解 C#语言，本节先来展现使用 VS 创建 C#应用程序。

(1) 启动 VS。

(2) 选择【文件】|【新建】|【项目】命令，打开【新建项目】对话框。

在【模板】| Visual C#下选择 Windows 选项，然后在中间窗格选择【控制台应用程序】选项。设置【名称】、【位置】和【解决方案名称】等，其他选项默认不变。设置如图 4-1 所示。

(3) 单击【确定】按钮，生成项目文件。修改其中的 Program.cs 文件如下，加粗部分是新增加的语句，其余是集成环境自动生成的。

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication1
{
    class Program
    {
        protected int i = 10;//声明一个名为 i 的整型变量，在声明的同时赋值为 10
        static void Main(string[] args)
        {
            Program p = new Program();
            p.i = 20;//为变量 i 赋值
            Console.WriteLine(p.i);//读取变量 i 的值
            Console.ReadKey();
        }
    }
}
```

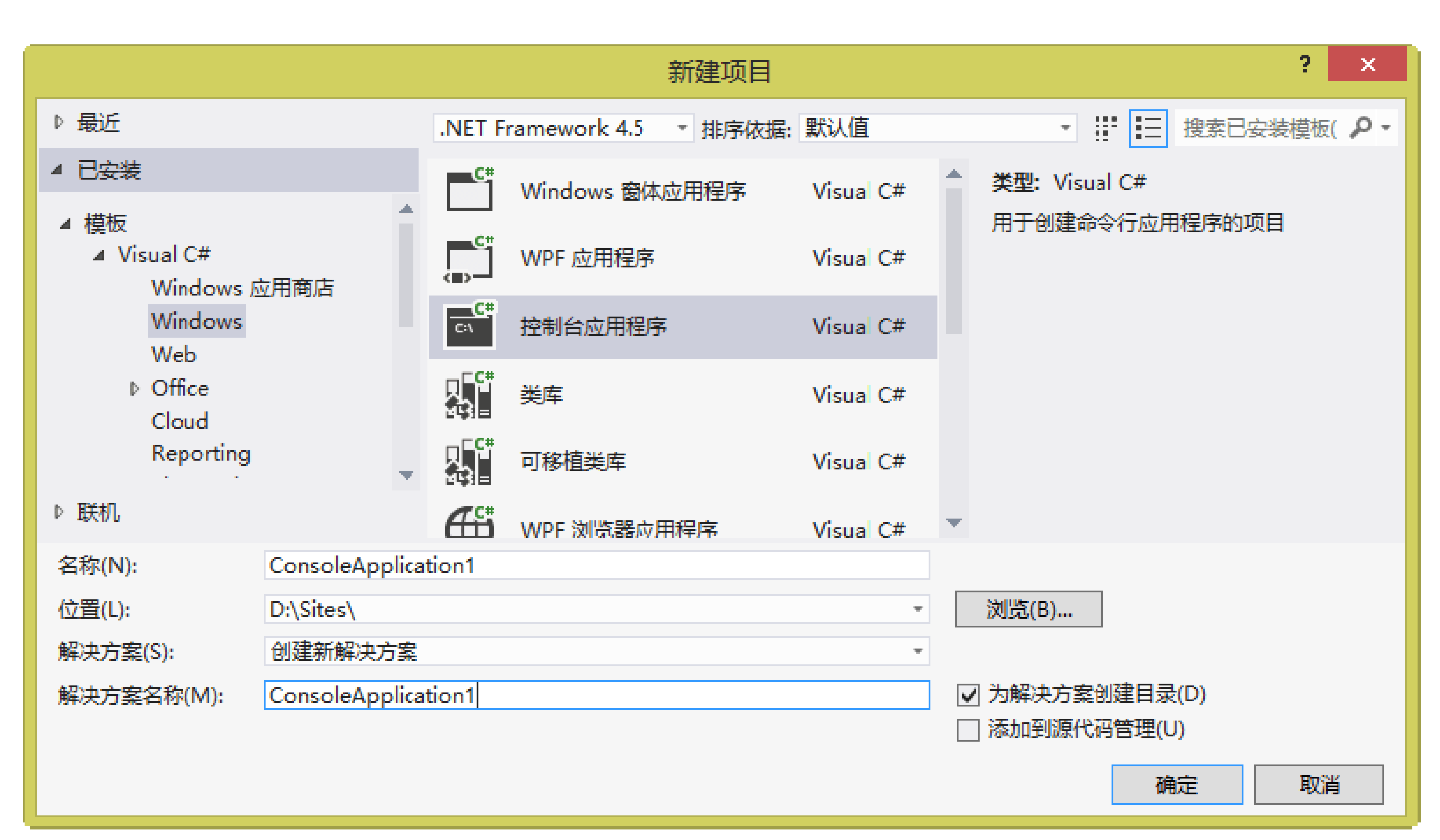


图 4-1　【新建项目】对话框

(4) 按 Ctrl+F5 组合键，运行程序，屏幕上出现一行字符，如图 4-2 所示，提示用户输入任意字符并按下 Enter 键返回。

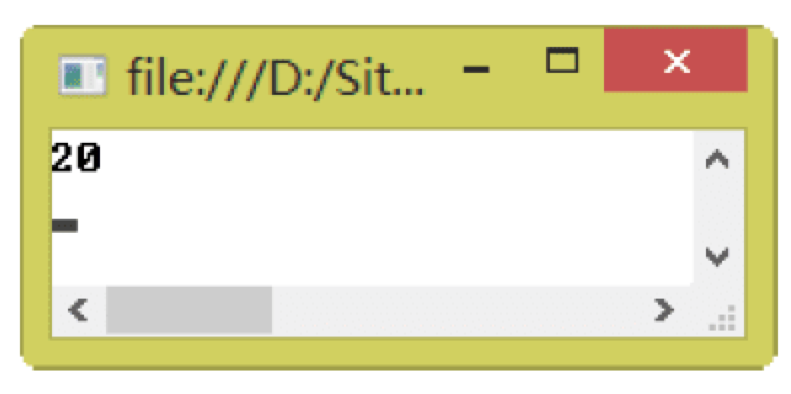


图 4-2　程序运行结果

4.3　类

C#语言是一种现代、面向对象的语言。面向对象程序设计方法提出了一个全新的概念：类。它的主要思想是将数据(数据成员)及处理这些数据的相应方法(函数成员)封装到类中，

类的实例则称为对象。这就是人们常说的封装性。

4.3.1　类的基本概念

类可以认为是对结构的扩充。类中不但可以包括数据，还包括处理这些数据的函数。类是对数据和处理数据的方法(函数)的封装。类是对某一类具有相同特性和行为的事物的描述。例如，定义一个描述个人情况的类 Person 如下。

```
using System;
class Person//类的定义，class 是保留字，表示定义一个类，Person 是类名
{
    private string name="张三";//类的数据成员声明
    private int age=12;//private 表示私有数据成员
    public void Display()//类的方法(函数)声明，显示姓名和年龄
    {
        Console.WriteLine("姓名:{0},年龄：{1}",name,age);
    }
    public void SetName(string PersonName)//修改姓名的方法(函数)
    {
        name=PersonName;
    }
    public void SetAge(int PersonAge)
    {
        age=PersonAge;
    }
}
```

Console.WriteLine("姓名:{0},年龄:{1}",name,age)的意义是将第二个参数变量 name 变为字符串填到{0}位置，将第三个参数变量 age 变为字符串填到{1}位置，将第一个参数表示的字符串在显示器上输出。

大家注意，这里实际定义了一个新的数据类型，为用户自己定义的数据类型，是对个人的特性和行为的描述，它的类型名为 Person，和 int、char 等一样，是一种数据类型。用定义的新数据类型 Person 类的方法把数据和处理数据的函数封装起来。类的声明格式如下：

```
属性 类修饰符 class 类名{类体}
```

其中，关键字 class、类名和类体是必需的，其他项是可选项。类修饰符包括 new、public、protected、internal、private、abstract 和 sealed，这些类修饰符在后面介绍。类体用于定义类的成员。

4.3.2　类成员的存取控制

一般希望类中一些数据不被随意修改，只能按指定方法修改，即隐蔽一些数据。同样，一些函数也不希望被其他类程序调用，只能在类内部使用。如何解决这个问题呢？可用访问权限控制符来实现，常用的访问权限控制符有 private(私有)，public(公有)。在数据成员或函数成员前增加访问权限控制符，可以指定该数据成员或函数成员的访问权限。

私有数据成员只能被类内部的函数使用和修改，私有函数成员只能被类内部的其他函数调用。类的公有函数成员可以被类的外部程序调用，类的公有数据成员可以被类的外部程序直接使用修改。公有函数实际是一个类和外部通信的接口，外部函数通过调用公有函数，按照预先设定好的方法修改类的私有成员。对于上述例子，name 和 age 是私有数据成员，只能通过公有函数 SetName()和 SetAge()修改。

4.3.3 类的对象

Person 类相当于是用户定义的一个新数据类型，由它可以生成 Person 类的实例，即对象。用如下方法声明类的对象。

```
Person OnePerson=new Person();
```

此语句的功能是建立 Person 类的对象，返回对象地址，赋值给 Person 类变量 OnePerson。也可以分两步创建 Person 类的对象，代码如下。

```
Person OnePerson;
OnePerson=new Person();
```

OnePerson 虽然存储的是 Person 类对象地址，但不是 C 中的指针，不能像指针那样可以进行加减运算，也不能转换为其他类型地址，它是引用型变量，只能引用(代表)Person 对象。

在程序中，可以用 OnePerson.方法名或 OnePerson.数据成员名访问对象的成员。例如：OnePerson.Display()，公用数据成员也可以这样访问。

4.3.4 类的构造函数和析构函数

在建立类的对象时，需做一些初始化工作，如对数据成员初始化。这些可以用构造函数来完成。每当用 new 生成类的对象时，自动调用类的构造函数。因此，可以把初始化的工作放到构造函数中完成。构造函数和类名相同，没有返回值。例如，可以定义 Person 类的构造函数如下。

```
public Person(string Name,int Age)//类的构造函数，函数名和类同名，无返回值。
{
    name=Name;
    age=Age;
}
```

当用 Person OnePerson=new Person("李四",16)语句生成 Person 类对象时，将自动调用以上构造函数。

变量和类的对象都有生命周期，生命周期结束，这些变量和对象就要被撤销。类的对象被撤销时，将自动调用析构函数。一些善后工作可放在析构函数中完成。析构函数的名字为类名，无返回类型，也无参数。Person 类的析构函数为 Person()。C#中类析构函数不能显示地被调用，它是被垃圾收集器撤销不被使用的对象时自动调用的。

4.3.5　类的构造函数的重载

在 C#语言中，同一个类中的函数，如果函数名相同，而参数类型或个数不同，认为是不同的函数，这叫函数重载。仅返回值不同，不能看作不同的函数。这样，可以在类定义中，定义多个构造函数，名字相同，参数类型或个数不同。根据生成类的对象方法不同，调用不同的构造函数。例如，可以定义 Person 类没有参数的构造函数如下。

```
public Person()//类的构造函数，函数名和类同名，无返回值。
{
    name="x";
    age=12;
}
```

用语句 Person OnePerson=new Person("李四",30)生成对象时，将调用有参数的构造函数，而用语句 Person OnePerson=new Person()生成对象时，调用无参数的构造函数。由于析构函数无参数，因此，析构函数不能重载。

4.3.6　一个使用类的例子

下边用一个完整的例子说明 Person 类的使用。

```
using System;
namespace e1//定义以下代码所属命名空间，意义见以后章节
{
    class Person
    {
        private String name="张三";//类的数据成员声明
        private int age=12;
        public void Display()//类的方法(函数)声明，显示姓名和年龄
        {
            Console.WriteLine("姓名:{0},年龄：{1}",name,age);
        }
        public void SetName(string PersonName)//指定修改姓名的方法(函数)
        {
            name=PersonName;
        }
        public void SetAge(int PersonAge)//指定修改年龄的方法(函数)
        {
            age=PersonAge;
        }
        public Person(string Name,int Age)//构造函数,函数名和类同名,无返回值
        {
            name=Name;
            age=Age;
        }
        public Person()//类的构造函数重载
        {
            name="田七";
            age=12;
        }
```

```
        }
        class Class1
        {
            static void Main(string[] args)
            {
                Person OnePerson=new Person("李四",30);//生成类的对象
                OnePerson.Display();
                //下句错误，在其他类(Class1 类)中，不能直接修改 Person 类中的私有成员。
                //OnePerson.name="王五";
                //只能通过 Person 类中公有方法 SetName 修改 Person 类中的私有成员 name。
                OnePerson.SetName("王五");
                OnePerson.SetAge(40);
                OnePerson.Display();
                OnePerson=new Person();
                OnePerson.Display();
            }
        }
    }
```

键入 CTRL+F5 运行后，显示的效果如下。

```
姓名: 李四,年龄：30
姓名: 王五,年龄：40
姓名: 田七,年龄：12
```

4.4　C#的数据类型

从大的方面来分，C#语言的数据类型可以分为 3 种：值类型、引用类型和指针类型。指针类型仅用于非安全代码中。本节重点讨论值类型和引用类型。

4.4.1　值类型和引用类型的区别

在 C#语言中，值类型变量存储的是数据类型所代表的实际数据，值类型变量的值(或实例)存储在栈(Stack)中，赋值语句是传递变量的值。引用类型(例如，类就是引用类型)的实例，也叫对象，不存在栈中，而存储在可管理堆(Managed Heap)中，堆实际上是计算机系统中的空闲内存。引用类型变量的值存储在栈(Stack)中，但存储的不是引用类型对象，而是存储引用类型对象的引用，即地址，和指针所代表的地址不同，引用所代表的地址不能被修改，也不能转换为其他类型地址，它是引用型变量，只能引用指定类对象，引用类型变量赋值语句是传递对象的地址。例如，以下示例代码。

```
using System;
class MyClass//类为引用类型
{
    public int a=0;
}
class Test
```

```
    {
        static void Main()
        {
            f1();
        }
        static public void f1()
        {
            int v1=1;//值类型变量 v1，其值 1 存储在栈(Stack)中
            int v2=v1;//将 v1 的值(为 1)传递给 v2，v2=1,v1 值不变。
            v2=2;//v2=2,v1 值不变。
            MyClass r1=new MyClass();//引用变量 r1 存储 MyClass 类对象的地址
            MyClass r2=r1;//r1 和 r2 都代表是同一个 MyClass 类对象
            r2.a=2;//和语句 r1.a=2 等价
        }
    }
```

存储在栈中的变量，当其生命周期结束，自动被撤销。例如，v1 存储在栈中，v1 和函数 f1 同生命周期，退出函数 f1，v1 不存在了。但在堆中的对象不能自动被撤销。.NET 系统 CLR 内建垃圾收集器，当对象的引用变量被撤销，表示对象的生命周期结束，垃圾收集器负责收回不被使用的对象占用的存储空间。例如，上例中引用变量 r1 及 r2 是 MyClass 类对象的引用，存储在栈中，退出函数 f1，r1 和 r2 都不存在了，在堆中的 MyClass 类对象也就被垃圾收集器撤销。

4.4.2　值类型变量分类

C#语言值类型可以分为以下几种。

1. 简单类型(Simple types)

简单类型中包括：数值类型和布尔类型(bool)。数值类型又细分为：整数类型、字符类型(char)、浮点数类型和十进制类型(decimal)。

2. 结构类型(Struct types)

3. 枚举类型(Enumeration types)

C#语言值类型变量无论如何定义，总是值类型变量，不会变为引用类型变量。

4.4.3　结构类型

结构类型可以声明构造函数、数据成员、方法、属性等。结构和类的最根本的区别是结构是值类型，类是引用类型。结构和类的区别是：结构不能从另外一个结构或者类派生，本身也不能被继承，因此不能定义抽象结构；结构成员也不能被访问权限控制字 protected 修饰，也不能用 virtual 和 abstract 修饰结构方法；在结构中不能定义析构函数。

虽然结构不能从类和结构派生，可是结构能够继承接口。结构继承接口的方法和类继承接口的方法基本一致。例如，下面的示例。

```
using System;
struct point//结构定义
{
    public int x,y;//结构中也可以声明构造函数和方法，变量不能赋初值
}
class Test
{
    static void Main()
    {
        point P1;
        P1.x=166;
        P1.y=111;
        point P2;
        P2=P1;//值传递，使 P2.x=166,P2.y=111
        point P3=new point();//用 new 生成结构变量 P3，P3 仍为值类型变量
        }//用 new 生成结构变量 P3 仅表示调用默认构造函数，使 x=y==0。
}
```

4.4.4　简单类型

简单类型也是结构类型，有构造函数、数据成员、方法、属性等，因此下列语句 int i=int.MaxValue;string s=i.ToString()是正确的。即使一个常量，C#也会生成结构类型的实例，因此也可以使用结构类型的方法。例如，string s=13.ToString()是正确的。

简单类型包括：整数类型、字符类型、布尔类型、浮点数类型、十进制类型，如表 4-1 所示。

表 4-1　简单类型

保留字	System 命名空间中的名字	字节数	取值范围
sbyte	System.Sbyte	1	-128~127
byte	System.Byte	1	0~255
short	System.Int16	2	-32768~32767
ushort	System.UInt16	2	0~65535
int	System.Int32	4	-2147483648~2147483647
uint	System.UInt32	4	0~4292967295
long	System.Int64	8	-9223372036854775808~9223372036854775808
ulong	System.UInt64	8	0~18446744073709551615
char	System.Char	2	0~65535
float	System.Single	4	3.4E-38~3.4E+38
double	System.Double	8	1.7E-308~1.7E+308
bool	System.Boolean		(true,false)
decimal	System.Decimal	16	正负 $1.0 \cdot 10^{-28}$ 到 $7.9 \cdot 10^{28}$之间

4.4.5　枚举类型

C#枚举类型的使用方法如下。

```
using System;
class Class1
{
      enum Days {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};
      //使用 Visual Studio.Net,enum 语句添加在[STAThread]前边
      static void Main(string[] args)
      {
         Days day=Days.Tue;
         int x=(int)Days.Tue;//x=2
         Console.WriteLine("day={0},x={1}",day,x);//显示结果为:day=Tue,x=4
      }
}
```

在此枚举类型 Days 中，每个元素的默认类型为 int，其中 Sun=0，Mon=1，Tue=2，依此类推。也可以直接给枚举元素赋值。如下所示。

```
enum Days{Sat=1,Sun,Mon,Tue,Wed,Thu,Fri,Sat};
```

在此枚举中，Sun=1，Mon=2，Tue=3，Wed=4，等等。C#枚举元素类型可以是 byte、sbyte、short、ushort、int、uint、long 和 ulong 类型，但不能是 char 类型。示例代码如下。

```
enum Days:byte{Sun,Mon,Tue,Wed,Thu,Fri,Sat};//元素为字节类型
```

4.4.6　值类型的初值和默认构造函数

所有变量都要求必须有初值，如没有赋值，采用默认值。对于简单类型，sbyte、byte、short、ushort、int、uint、long 和 ulong 默认值为 0，char 类型默认值是(char)0，float 为 0.0f，double 为 0.0d，decimal 为 0.0m，bool 为 false，枚举类型为 0，在结构类型和类中，数据成员的数值类型变量设置为默认值，引用类型变量设置为 null。

可以显示的赋值，例如 int i=0。而对于复杂结构类型，其中的每个数据成员都按此种方法赋值，显得过于麻烦。由于数值类型都是结构类型，可用 new 语句调用其构造函数初始化数值类型变量，如 int j=new int()。请注意，用 new 语句并不是把 int 变量变为引用变量，j 仍是值类型变量，这里 new 仅仅是调用其构造函数。所有的数值类型都有默认的无参数的构造函数，其功能就是为该数值类型赋初值为默认值。对于自定义结构类型，由于已有默认的无参数的构造函数，不能再定义无参数的构造函数，但可以定义有参数的构造函数。

4.4.7　引用类型分类

C#语言中引用类型可以分为以下几种。

- 类：C#语言中预定义了对象类(object 类)、数组类、字符串类等。当然，开发人员也可以定义其他类。
- 接口。
- 代表。

C#语言引用类型变量无论如何定义，总是引用类型变量，不会变为值类型变量。C#语言引用类型对象一般用运算符 new 建立，用引用类型变量引用该对象。本节仅介绍对象类型(object 类型)、字符串类型、数组。

4.4.8　对象类(object 类)

C#中的所有类型(包括数值类型)都直接或间接地以 object 类为基类。对象类(object 类)是所有其他类的基类。任何一个类定义，如果不指定基类，默认 object 为基类。继承和基类的概念见后续章节。C#语言规定，基类的引用变量可以引用派生类的对象，因此，对一个 object 的变量可以赋予任何类型的值。

```
int x =25;
object obj1;
obj1=x;
object obj2= 'A';
```

object 关键字是在命名空间 System 中定义的，是类 System.Object 的别名。

4.4.9　数组类

数组是一组类型相同的有序数据。数组按照数组名、数据元素的类型和维数来进行描述。C#语言中数组是类 System.Array 类对象，如声明一个整型数数组：int[] arr=new int[5]; 实际上生成了一个数组类对象，arr 是这个对象的引用(地址)。

在 C#中数组可以是一维的也可以是多维的，同样也支持数组的数组，即数组的元素还是数组。一维数组最为普遍，用的也最多。如下所示。

```
using System;
class Test
{
    static void Main()
    { int[] arr=new int[3];//用 new 运算符建立一个 3 个元素的一维数组
        for(int i=0;i<arr.Length;i++)//arr.Length 是数组类变量，表示数组元素个数
            arr[i]=i*i;//数组元素赋初值，arr[i]表示第 i 个元素的值
        for (int i=0;i<arr.Length;i++)//数组第一个元素的下标为 0
            Console.WriteLine("arr[{0}]={1}",i,arr[i]);
    }
}
```

这个程序创建了一个 int 类型 3 个元素的一维数组，初始化后逐项输出。其中 arr.Length 表示数组元素的个数。程序的输出如下。

```
arr[0] = 0
arr[1] = 1
arr[2] = 4
```

上面的例子中使用的是一维数组，下面介绍多维数组。示例如下。

```
string[] a1;//一维 string 数组类引用变量 a1
```

```
string[,] a2;//二维 string 数组类引用变量 a2
a2=new string[2,3];
a2[1,2]="abc";
string[,,] a3;//三维 string 数组类引用变量 a3
string[][] j2;//数组的数组，即数组的元素还是数组
string[][][][] j3;
在数组声明的时候，可以对数组元素进行赋值。看下面的例子：
int[] a1=new int[]{1,2,3};//一维数组，有 3 个元素。
int[] a2=new int[3]{1,2,3};//此格式也正确
int[] a3={1,2,3};//相当于 int[] a3=new int[]{1,2,3};
int[,] a4=new int[,]{{1,2,3},{4,5,6}};//二维数组，a4[1,1]=5
int[][] j2=new int[3][];//定义数组 j2,有三个元素，每个元素都是一个数组
j2[0]=new int[]{1,2,3};//定义第一个元素，是一个数组
j2[1]=new int[]{1, 2, 3, 4, 5, 6};//每个元素的数组可以不等长
j2[2]=new int[]{1, 2, 3, 4, 5, 6, 7, 8, 9};
```

4.4.10　字符串类(string 类)

C#还定义了一个基本的类 string，专门用于对字符串的操作。这个类也是在命名空间 System 中定义的，是类 System.String 的别名。

1.　字符串定义

字符串定义示例代码如下。

```
string s;//定义一个字符串引用类型变量 s
s="Zhang";//字符串引用类型变量 s 指向字符串"Zhang"
string   FirstName="Ming";
string   LastName="Zhang";
string   Name=FirstName+"   "+LastName;//运算符+已被重载
string   SameName=Name;
char[] s2={'计','算','机','科','学'};
string s3=new string(s2);
```

2. 字符串搜索

字符串搜索的示例代码如下。

```
string s="ABC 科学";
int i=s.IndexOf("科");
```

搜索"科"在字符串中的位置，因第一个字符索引为 0，所以"A"索引为 0，"科"索引为 3，因此这里 i=3，如没有此字符串 i=-1。注意 C#中，ASCII 和汉字都用 2 字节表示。

3. 字符串比较函数

字符串比较函数使用的示例代码如下。

```
string s1="abc";
string s2="abc";
int n=string.Compare(s1,s2);//n=0
```

n=0 表示两个字符串相同，n 小于零，s1<s2，n 大于零，s1>s2。此方法区分大小写。也可用如下办法比较字符串。

```
string s1="abc";
string s="abc";
string s2="不相同";
```

if(s==s1)//还可用!=。虽然 String 是引用类型，但这里比较两个字符串的值。

```
s2="相同";
```

4. 判断是否为空字符串

判断是否为空字符串的示例代码如下。

```
string s="";
string s1="不空";
if(s.Length==0)
s1="空";
```

5. 得到子字符串或字符

获取子字符串或字符的示例代码如下。

```
string s="取子字符串";
string sb=s.Substring(2,2);//从索引为 2 开始取 2 个字符，Sb="字符",s 内容不变
char sb1=s[0];//sb1='取'
Console.WriteLine(sb1);//显示：取
```

6. 字符串删除函数

字符串删除函数的使用示例如下。

```
string s="取子字符串";
string sb=s.Remove(0,2);//从索引为 0 开始删除 2 个字符，Sb="字符串",s 内容不变
```

7. 插入字符串

插入字符串的示例代码如下。

```
string s="计算机科学";
string s1=s.Insert(3,"软件");//s1="计算机软件科学"，s 内容不变
```

8. 字符串替换函数

字符串替换函数的使用示例如下。

```
string s="计算机科学";
string s1=s.Replace("计算机","软件");//s1="软件科学"，s 内容不变
```

9. 把 String 转换为字符数组

把 String 对象转换为字符数组的示例如下。

```
string S="计算机科学";
```

```
char[] s2=S.ToCharArray(0,S.Length);//属性 Length 为字符类对象的长度
```

10. 其他数据类型转换为字符串

以下是一些其他数据类型转换的示例。

```
int i=9;
string s8=i.ToString();//s8="9"
float n=1.9f;
string s9=n.ToString();//s8="1.9"
```

其他数据类型都可用此方法转换为字符类对象。

11. 大小写转换

大小写转换的示例代码如下。

```
string s="AaBbCc";
string s1=s.ToLower();//把字符转换为小写，s 内容不变
string s2=s.ToUpper();//把字符转换为大写，s 内容不变
```

12. 删除所有的空格

删除字符串中的空格示例代码如下。

```
string s="A     bc ";
s.Trim();//删除所有的空格
```

string 类其他方法的使用请用帮助系统查看，方法是打开 Visual Studio.Net 的代码编辑器，键入 string，将光标移到键入的字符串 string 上，然后按 F1 键。

4.4.11　类型转换

在编写 C#语言程序中，经常会碰到类型转换问题。例如，整型数和浮点数相加，C#会进行隐式转换。C#语言中类型转换分为：隐式转换、显示转换、加框(boxing)和消框(unboxing)等 3 种。

1. 隐式转换

隐式转换就是系统默认的、不需要加以声明就可以进行的转换。例如，从 int 类型转换到 long 类型就是一种隐式转换。在隐式转换过程中，转换一般不会失败，转换过程中也不会导致信息丢失。示例代码如下。

```
int j=10;
long i=j;
```

2. 显示转换

显式类型转换，又叫强制类型转换。与隐式转换正好相反，显式转换需要明确地指定转换类型，显示转换可能导致信息丢失。下面的例子把长整形变量显式转换为整型。

```
long J=1000;
int I=(int)J;//如果超过 int 取值范围，将产生异常
```

3. 加框(boxing)和消框(unboxing)

加框(boxing)和消框(unboxing)是 C#语言类型系统提出的核心概念，加框是值类型转换为 object(对象)类型，消框是 object(对象)类型转换为值类型。有了加框和消框的概念，对任何类型的变量来说最终都可以看作是 object 类型。

(1) 加框操作

把一个值类型变量加框也就是创建一个 object 对象，并将这个值类型变量的值复制给这个 object 对象。示例代码如下。

```
int i=10;
object obj=i;//隐式加框操作，obj 为创建的 object 对象的引用。
```

也可以用显式的方法来进行加框操作，示例代码如下。

```
int i =10;
object obj=object(i);//显式加框操作
```

值类型的值加框后，值类型变量的值不变，仅将这个值类型变量的值复制给这个 object 对象。例如，下面的程序。

```
using System;
class Test
{
    public static void Main()

    {
        int n=100;
        object o=n;
        o=300;//不能改变 n
        Console.WriteLine("{0},{1}",n,o);
    }
}
```

输出结果为：100，300。这就证明了值类型变量 n 和 object 类对象 o 都独立存在着。

(2) 消框操作

和加框操作正好相反，消框操作是指将一个对象类型显式地转换成一个值类型。消框的过程分为两步：首先检查这个 object 对象，看它是否为给定的值类型的加框值，如是，把这个对象的值拷贝给值类型的变量。示例代码如下。

```
int i=10;
object obj=i;
int j=(int)obj;//消框操作
```

可以看出消框过程正好是加框过程的逆过程，必须注意加框操作和消框操作必须遵循类型兼容的原则。

(3) 加框和消框的使用

定义如下函数。

```
void Display(Object o)//注意，o 为 Object 类型
{
  int x=(int)o;//消框
  System.Console.WriteLine("{0},{1}",x,o);
}
```

调用此函数：int y=20;Display(y);在此利用了加框概念。虚参被实参替换：Object o=y，也就是说，函数的参数是 Object 类型，可以将任意类型实参传递给函数。

4.5　运算符

C#语言和 C 语言的运算符用法基本一致。本节主要简单介绍 C#语言中的运算符。

4.5.1　运算符分类

按照操作数的个数，C#语言的运算符可以分为以下几种类型。

- 一元运算符：一元运算符作用于一个操作数，如：-X、++X、X--等。
- 二元运算符：二元运算符对两个操作数进行运算，如 x+y。
- 三元运算符：三元运算符只有一个 x? y:z。

C#语言运算符的详细分类及操作符从高到低的优先级顺序如表 4-2 所示。

表 4-2　运算符

类　别	操　作　符
初级操作符	(x) x.y f(x) a[x] x++ x-- new type of sizeof checked unchecked
一元操作符	+ - ! ~ ++x –x (T)x
乘除操作符	* / %
加减操作符	+ -
移位操作符	<< >>
关系操作符	< > <= >= is as
等式操作符	== !=
逻辑与操作符	&
逻辑异或操作符	^
逻辑或操作符	|
条件与操作符	&&
条件或操作符	||
条件操作符	?:
赋值操作符	= *= /= %= += -= <<= >>= &= ^= |=

4.5.2　测试运算符 is

is 操作符用于动态地检查表达式是否为指定类型。使用格式为：e is T，其中 e 是一个表达式，T 是一个类型，该式判断 e 是否为 T 类型，返回值是一个布尔值。示例代码如下。

```
using System;
class Test
{
    public static void Main()
    {
        Console.WriteLine(1 is int);
        Console.WriteLine(1 is float);
        Console.WriteLine(1.0f is float);
        Console.WriteLine(1.0d is double);
    }
}
```

输出为：

```
True
False
True
True
```

4.5.3　typeof 运算符

typeof 操作符用于获得指定类型在 system 命名空间中定义的类型名字，示例代码如下。

```
using System;
class Test
{
    static void Main()
    {
        Console.WriteLine(typeof(int));
        Console.WriteLine(typeof(System.Int32));
        Console.WriteLine(typeof(string));
        Console.WriteLine(typeof(double[]));
    }
}
```

产生如下输出，由输出可知 int 和 System.int32 是同一类型。

```
System.Int32
System.Int32
System.String
System.Double[]
```

4.5.4　溢出检查操作符 checked 和 unchecked

在进行整型算术运算(如+、-、*、/等)或从一种整型显式转换到另一种整型时，有可能

出现运算结果超出这个结果所属类型值域的情况，这种情况称之为溢出。整型算术运算表达式可以用 checked 或 unchecked 溢出检查操作符，决定在编译和运行时是否对表达式溢出进行检查。如果表达式不使用溢出检查操作符或使用了 checked 操作符，常量表达式溢出，在编译时将产生错误，表达式中包含变量，程序运行时执行该表达式产生溢出，将产生异常提示信息。而使用了 unchecked 操作符的表达式语句，即使表达式产生溢出，编译和运行时都不会产生错误提示。示例代码如下。

```
using System;
class Class1
{
    static void Main(string[] args)
    {
        const int x=int.MaxValue;
        unchecked//不检查溢出
        {
            int z=x*2;//编译时不产生编译错误，z=-2
            Console.WriteLine("z={0}",z);//显示-2
        }
        checked//检查溢出
        {
            int z1=(x*2);//编译时会产生编译错误
            Console.WriteLine("z={0}",z1);
        }
    }
}
```

4.5.5　new 运算符

new 操作符可以创建值类型变量、引用类型对象，同时自动调用构造函数。示例代码如下。

```
int y=new int();//用 new 创建整型变量 y，调用默认构造函数
Person p1=new Person ();//用 new 建立的 Person 类对象。Person 变量 p1 对象的引用
int[] arr=new int[2];//数组也是类，创建数组类对象，arr 是数组对象的引用
```

需注意的是，int y=new int()语句将自动调用 int 结构不带参数的构造函数，给 y 赋初值 0，y 仍是值类型变量，不会变为引用类型变量。

4.5.6　运算符优先级

当一个表达式包含多种操作符时，操作符的优先级控制着操作符求值的顺序。例如，表达式x+y*z按照x+(y*z)顺序求值，因为*操作符比+操作符有更高的优先级。这和数学运算中的先乘除后加减是一致的。

当两个有相同优先级的操作符对操作数进行运算时，如 x+y-z，操作符按照出现的顺序由左至右执行，x+y-z 按(x+y)-z 进行求值。赋值操作符按照右接合的原则，即操作按照从右向左的顺序执行，如 x=y=z 按照 x=(y=z)进行求值。

4.6　流程控制语句

在实际开发中，会涉及非常复杂的业务逻辑跳转或流转跳转的功能。开发此类功能时，就需要开发人员根据外部条件来判断程序的走向，从而使程序能够智能化地完成客户所需的工作。C#中提供了很多流程控制语句，来帮助开发人员完成此类工作，本节将介绍 C#中常见的流程控制语句，希望读者能迅速、熟练掌握这些流程控制语句，从而完成复杂的业务逻辑开发。

4.6.1　if 语句

实际开发中经常需要根据用户输入的值或者用户的选择来判断程序的走向。C#提供了 if 语句，用于进行判断并控制程序走向，其一般形式如下。

```
if (条件判断表达式)
 {
  //当条件表达式为真时执行的代码段
  }
```

例如，判断学生年龄是否大于 18 岁，如果大于就输出“该学生已成年”的消息，可以使用以下代码。

```
static void Main(string[] args)
{
    int age = 0;
    Console.WriteLine("请输入学生的年龄");
    age = int.Parse(Console.ReadLine());
    //判断学生年龄是否大于 18 岁，若是，则输出
    if (age >= 18)
    {
        Console.WriteLine("该学生已成年");
    }
}
```

在以上代码段中，如果输入的年龄大于或等于 18，则执行 if 语句块中的语句，否则程序将不执行。

另外，if 语句还拥有 if…else 这样的扩展形式，语法格式如下。

```
if (条件判断表达式)
{
  //当条件判断表达式为真时执行的代码块
}else{
  //当条件判断表达式为假时执行的代码块
}
```

例如，前面的代码可改写成如下形式，若学生未满 18 岁时，输出“该学生未成年”。

```
if (age >= 18)
```

```
{
    Console.WriteLine("该学生已成年");
}else{
      Console.WriteLine("该学生未成年");
}
```

除此之外，还可以将以上两种语法格式进行更复杂的嵌套，如 if…else if…else 形式，这样的扩展形式拥有多个条件表达式判断分支，根据条件表达式控制程序执行不同分支中的语句块。例如，以下代码根据用户的年龄输出用户的年龄层次时，使用了这种语句形式进行多分支判断。

```
if (age >= 18 && age < 30)
{
     Console.WriteLine("青年");
}
else if (age >= 30 && age < 45)
{
    Console.WriteLine("中青年");
}
else if (age >= 45 && age < 55)
{
     Console.WriteLine("中年");
}
    else if (age >= 55)
{
    Console.WriteLine("老年");
}
else
{
     Console.WriteLine("未成年");
}
```

4.6.2　switch 语句

Swithch 语句是一种特殊的分支语句，其一般形式如下。

```
switch(变量)
 {
case  值 1:
   代码块;
   break;
 case  值 2:
    代码块;
     break;
…
 default:
    代码块;
     break;
}
```

switch 语句具有以下特点：

- 该语句进行分支判断时，只进行简单的等值判断，即传入 switch 语句中的变量值与某一个 case 语句中的值相等时，将执行该 case 语句下的语句块。
- 该语句块中的 break 关键字可以省略，但是当某 case 语句块中没有 break 关键字时，该 case 语句块中不能有任何代码。
- default 语句块是可选的，当传入 switch 语句块中的变量不等于任何一个 case 语句中的值时，将执行 default 语句块中的代码。
- 当该语句中的任一 case 或 default 代码块被执行后，将跳出 switch 代码块执行之后的代码。

如果判断某个数是否为 10 以内的质数，可以使用以下代码。

```
static void Main(string[] args)
{
    int i = 0;
    Console.WriteLine("请输入一个 1-10 之间的数字：");
    i = int.Parse(Console.ReadLine());
    switch(i)
    {
        case 2:
        case 3:
        case 5:
        case 7: //只要传入的变量值为以上 case 中的任何一个，即执行 case 中的代码块
          Console.WriteLine("该数是一个质数");
          break;
          default:
          Console.WriteLine("该数不是一个 1-10 之间的质数");
          break;
    }

}
```

以上代码执行时，当变量 i 的值等于 2、3、5、7 中的任何值时，就会执行 case 7 中的代码块，否则会执行 default 代码块中的语句。

4.6.3　for 语句

C#提供了 for 语句进行循环控制，其一般形式如下。

```
for (循环变量初始化表达式;循环条件表达式;循环变量变更表达式)
{
    //当循环变量满足循环条件时执行的代码块
}
```

for 语句具有以下特点。

- 该语句中的 3 个表达式都是可以省略的，但是，如果 3 个表达式都被省略，则会形成死循环。

- 该语句第一次循环时，首先根据循环变量初始化表达式及循环变量，然后判断变量是否满足循环条件表达式，如果满足则执行循环，否则终止，这样的过程被称为一次迭代过程；第一次迭代完成后，for 语句将执行循环变量变更表达式，变更循环变量的值，再次判断变量是否满足循环条件表达式，如果满足则执行循环，否则终止循环。
- 该语句会根据循环条件表达式的返回值，决定是否继续执行循环体中的代码。当循环条件表达式的返回值为 true 时，则继续执行循环体中的代码，否则将终止循环。

例如，下面的代码使用 for 循环，计算出 1-100 之间的所有偶数。

```
static void Main(string[] args)
{
        //for (循环变量初始化表达式;循环条件表达式;循环变量变更表达式)
        for(int i=1;i<=100;i++)
        {
                if(i%2==0)
                {
                        Console.Write(i+"\t");
                }
        }
}
```

4.6.4　foreach 语句

C#中拥有多种用于存储数据的集合，在早期的一些语言中都使用 for 循环获取集合中的元素。C#中也提供了 foreach 语句，用于循环集合中的元素，其一般形式使用如下形式。

```
foreach(元素类型  集合中的元素变量  in  集合)
```

需要注意的是，foreach 语句只可以获取到集合中元素的值，但是不能对集合中的元素值进行更改，否则在运行时会抛出异常。例如，可以使用该语句循环遍历数组中的元素，代码如下。

```
static void Main(string[] args)
{
   int[] arr = { 1, 3, 5, 7, 9 };
   foreach (int i in arr)
{
      Console.Write(i + "\t");
}
      Console.WriteLine();
}
```

以上代码中的 foreach 语句循环了数组中的所有元素，当循环至最后一个元素时，foreach 会自动退出。

4.6.5　while 语句

C#提供了一种简单易用的循环语句，即 while 语句，其一般形式如下。

```
while(条件表达式)
{
 //循环执行的语句
}
```

while 语句仅需要条件表达式，如果为了防止形成死循环，还需要在 while 循环中添加循环变量变更语句。例如，求 1-100 间的偶数，可以使用以下代码。

```
static void Main(string[] args)
{
      int i=1;               //循环变量初始化表达式
      while(i<=100)      //循环判断表达式
 {
        if(i%2==0)
           Console.write(i+"\t");
        i++;        //循环控制变量变更表达式
 }
}
```

通过以上代码，读者可以发现，实际上 while 循环也具有 3 个循环表达式，用于初始化循环变量、判断循环条件和变更循环变量。

while 循环一般用于进行迭代次数不定的循环，如读取文件流等。while 循环中也可以使用 continue、break 关键字进行循环控制。

4.7　类的继承

前面定义了一个 Person 类，如果需要定义一个雇员类，可以从头开始定义雇员类 Employee。但这样不能利用 Person 类中已定义的函数和数据。比较好的方法是，以 Person 类为基类，派生出一个雇员类 Employee，雇员类 Employee 继承了 Person 类的数据成员和函数成员，即 Person 类的数据成员和函数成员成为 Employee 类的成员。这个 Employee 类叫以 Person 类为基类的派生类，这是 C#提出的方法。C#用继承的方法，实现代码的重用。

4.7.1　派生类的声明格式

派生类的声明格式如下。

```
属性  类修饰符  class  派生类名:基类名  {类体}
```

雇员类 Employee 定义如下。

```
class Employee : Person//Person 类是基类
{
```

```
        private string department;//部门，新增数据成员
        private decimal salary;//薪金，新增数据成员
        public Employee(string Name, int Age, string D, decimal S): base(Name, Age)
        {//注意 base 的第一种用法，根据参数调用指定基类构造函数，注意参数的传递
            department = D;
            salary = S;
        }
        public new void Display()//覆盖基类 Display()方法,注意 new,不可用 override
        {
            base.Display();//访问基类被覆盖的方法，base 的第二种用法
            Console.WriteLine("部门：{0}   薪金：{1}", department, salary);
        }
    }
```

修改主函数如下。

```
    class Class1
    {
        static void Main(string[] args)
        {
            Employee OneEmployee = new Employee("李四", 30, "计算机系", 2000);
            OneEmployee.Display();
        }
    }
```

Employee 类继承了基类 Person 的方法 SetName()、SetAge()，数据成员 name 和 age，即认为基类 Person 的这些成员也是 Employee 类的成员，但不能继承构造函数和析构函数。添加了新的数据成员 department 和 salary。覆盖了方法 Display()。

虽然 Employee 类继承了基类 Person 的 name 和 age，但由于它们是基类的私有成员，Employee 类中新增或覆盖的方法不能直接修改 name 和 age，只能通过基类原有的公有方法 SetName()和 SetAge()修改。如果希望在 Employee 类中能直接修改 name 和 age，必须在基类中修改它们的属性为 protected。

4.7.2　base 关键字

base 关键字用于从派生类中访问基类成员，它有以下两种基本用法。

- 在定义派生类的构造函数中，指明要调用的基类构造函数，由于基类可能有多个构造函数，根据 base 后的参数类型和个数，指明要调用哪一个基类构造函数。如上节雇员类 Employee 构造函数定义中的 base 的第一种用法。
- 在派生类的方法中调用基类中被派生类覆盖的方法。如上节雇员类 Employee 的 Display()方法定义中的 base 的第二种用法。

4.7.3　覆盖基类成员

在派生类中，通过声明与基类完全相同新成员，可以覆盖基类的同名成员，完全相同是指函数类型、函数名、参数类型和个数都相同，如上例的 Display()方法。派生类覆盖基

类成员不算错误，但会导致编译器发出警告。如果增加 new 修饰符，表示认可覆盖，编译器不再发出警告。

4.7.4　C#语言类继承的特点

C#语言类继承有如下特点。

- C#语言只允许单继承，即派生类只能有一个基类。
- C#语言继承是可以传递的，如果 C 从 B 派生，B 从 A 派生，那么 C 不但继承 B 的成员，还要继承 A 中的成员。
- 派生类可以添加新成员，但不能删除基类中的成员。
- 派生类不能继承基类的构造函数、析构函数和事件。但能继承基类的属性。
- 派生类可以覆盖基类的同名成员，如果在派生类中覆盖了基类同名成员，基类该成员在派生类中就不能被直接访问，只能通过【base. 基类方法名】访问。
- 派生类对象也是其基类的对象，但基类对象却不是其派生类的对象。例如，前边定义的雇员类 Employee 是 Person 类的派生类，所有雇员都是人类，但很多人并不是雇员，可能是教师、学生、摄影师等。因此，基类的引用变量可以引用其派生类对象，但派生类的引用变量不可以引用其基类对象。

4.8　类的成员

由于 C#程序中每个变量或函数都必须属于一个类或结构，因此所有的变量或函数都是类或结构的成员。类的成员可以分为两大类：类本身所声明的以及从基类中继承来的。

4.8.1　类的成员类型

类的成员有局部变量、字段、方法成员、属性、事件、索引指示器、操作符重载、构造函数和析构函数。各成员类型说明如下。

- 局部变量：在 for、switch 等语句中和类方法中定义的变量，只在指定范围内有效。
- 字段：即类中的变量或常量，包括静态字段、实例字段、常量和只读字段。
- 方法成员：包括静态方法和实例方法。
- 属性：按属性指定的 get 方法和 Set 方法对字段进行读写。
- 事件：代表事件本身，同时联系事件和事件处理函数。
- 索引指示器：允许像使用数组那样访问类中的数据成员。
- 操作符重载：采用重载操作符的方法定义类中特有的操作。
- 构造函数和析构函数。

其中，包含有可执行代码的成员被认为是类中的函数成员，这些函数成员有方法、属性、索引指示器、操作符重载、构造函数和析构函数。

4.8.2　类成员访问修饰符

访问修饰符用于指定类成员的可访问性，C#访问修饰符有 4 种，分别是 private、protected、public 和 internal。

- private 用于声明私有成员，私有数据成员只能被类内部的函数使用和修改，私有函数成员只能被类内部的函数调用。派生类虽然继承了基类私有成员，但不能直接访问它们，只能通过基类的公有成员访问。
- protected 声明保护成员，保护数据成员只能被类内部和派生类的函数使用和修改，保护函数成员只能被类内部和派生类的函数调用。
- public 声明公有成员，类的公用函数成员可以被类的外部程序所调用，类的公用数据成员可以被类的外部程序直接使用。公有函数实际是一个类和外部通讯的接口，外部函数通过调用公有函数，按照预先设定好的方法修改类的私有成员和保护成员。
- internal 声明内部成员，内部成员只能在同一程序集中的文件中才是可以访问的，一般是同一个应用(Application)或库(Library)。

4.9　类的字段和属性

一般把类或结构中定义的变量和常量叫字段。属性不是字段，本质上是定义修改字段的方法。由于属性和字段的紧密关系，把它们放到一起叙述。

4.9.1　静态字段、实例字段、常量和只读字段

用修饰符 static 声明的字段为静态字段。不管包含该静态字段的类生成多少个对象或根本无对象，该字段都只有一个实例，静态字段不能被撤销。一般采用如下方法引用静态字段。

```
类名.静态字段名
```

如果类中定义的字段不使用修饰符 static，该字段为实例字段，每创建该类的一个对象，在对象内创建一个该字段实例，创建它的对象被撤销，该字段对象也被撤销。实例字段采用如下方法引用。

```
实例名.实例字段名
```

用 const 修饰符声明的字段为常量，常量只能在声明中初始化，以后不能再修改。用 readonly 修饰符声明的字段为只读字段，只读字段是特殊的实例字段，它只能在字段声明中或构造函数中重新赋值，在其他任何地方都不能改变只读字段的值。示例如下。

```
public class Test
{
```

```
        public const int intMax = int.MaxValue;//常量，必须赋初值
        public int x = 0;//实例字段
        public readonly int y = 0;//只读字段
        public static int cnt = 0;//静态字段
        public Test(int x1, int y1)//构造函数
        { //intMax=0;//错误，不能修改常量
            x = x1;//在构造函数允许修改实例字段
            y = y1;//在构造函数允许修改只读字段
            cnt++;//每创建一个对象都调用构造函数，用此语句可以记录对象的个数
        }
        public void Modify(int x1, int y1)
        { //intMax=0;//错误，不能修改常量
            x = x1;
            cnt = y1;
            //y=10;//不允许修改只读字段
        }
    }
    class Class1
    {
        static void Main(string[] args)
        {
            Test T1 = new Test(100, 200);
            T1.x = 40;//引用实例字段采用方法：实例名.实例字段名
            Test.cnt = 0;//引用静态字段采用方法：类名.静态字段名
            int z = T1.y;//引用只读字段
            z = Test.intMax;//引用常量
        }
    }
```

4.9.2　属性

C#语言支持组件编程。组件也是类，组件用属性、方法、事件描述。属性不是字段，但必然和类中的某个或某些字段相联系，属性定义了得到和修改相联系的字段的方法。C#中的属性更充分地体现了对象的封装性：不直接操作类的数据内容，而是通过访问器进行访问，借助于 get 和 set 方法对属性的值进行读写。访问属性值的语法形式和访问一个变量基本一样，使访问属性就像访问变量一样方便，符合用户习惯。

在类的基本概念一节中，定义一个描述人的 Person 类，其中字段 name 和 age 是私有字段，记录姓名和年龄，外部通过公有方法 SetName 和 SetAge 修改这两个私有字段。现在用属性来描述姓名和年龄。示例代码如下。

```
using System;
public class Person
{
 private string P_name="张三";//P_name 是私有字段
 private int P_age=12;//P_age 是私有字段
      public void Display()//类的方法声明，显示姓名和年龄
 {
        Console.WriteLine("姓名:{0},年龄：{1}",P_name,P_age);
```

```
    }
    public string Name//定义属性 Name
    {
          get
         {return    P_name;}
         set
         {P_name=value;}
    }
    public int Age//定义属性 Age
    {
         get
         {return    P_age;}
         set
         {P_age=value;}
    }
}
public class Test
{
      public static void Main()
      {
          Person OnePerson= new Person();
         OnePerson.Name="田七";//value="田七"，通过 set 方法修改变量 P_Name
         string s=OnePerson.Name;//通过 get 方法得到变量 P_Name 值
         OnePerson.Age=20;//通过定义属性，既保证了姓名和年龄按指定方法修改
         int x=OnePerson.Age;//语法形式和修改、得到一个变量基本一致，符合习惯
         OnePerson.Display();
      }
}
```

在属性的访问声明中，只有 set 访问器表明属性的值只能进行设置而不能读出，只有 get 访问器表明属性的值是只读的不能改写，同时具有 set 访问器和 get 访问器表明属性的值的读写都是允许的。

虽然属性和字段的语法比较类似，但由于属性本质上是方法，因此不能把属性当作变量那样使用，也不能把属性作为引用型参数或输出参数来进行传递。

4.10　类的方法

方法是类中用于执行计算或其他行为的成员。所有方法都必须定义在类或结构中。

4.10.1　方法的声明

方法的声明格式如下。

```
    属性  方法修饰符  返回类型  方法名(形参列表){方法体}
```

方法修饰符包括 new、public、protected、internal、private、static、virtual、sealed、override、abstract 和 extern。这些修饰符有些已经介绍过，其他修饰符将逐一介绍。返回类型可以是

任何合法的 C#数据类型，也可以是 void，即无返回值。形参列表的格式为：(形参类型 形参 1，形参类型 形参 2，…)，可以有多个形参。

4.10.2　方法参数的种类

C#语言的方法可以使用如下 4 种参数。

- 值参数，不含任何修饰符。
- 引用参数，以 ref 修饰符声明。
- 输出参数，以 out 修饰符声明。
- 数组参数，以 params 修饰符声明。

1. 值参数

当用值参数向方法传递参数时，程序给实参的值做一份拷贝，并且将此拷贝传递给该方法，被调用的方法不会修改实参的值，所以使用值参数时，可以保证实参的值是安全的。如果参数类型是引用类型，如类的引用变量，则拷贝中存储的也是对象的引用，所以拷贝和实参引用同一个对象。通过这个拷贝，可以修改实参所引用的对象中的数据成员。

2. 引用参数

有时在方法中，需要修改或得到方法外部的变量值，C#语言用引用参数向方法传递实参，程序将把实参的引用，即实参在内存中的地址传递给方法，方法通过实参的引用，修改或得到方法外部的变量值。引用参数以 ref 修饰符声明。注意，实参变量要求必须被设置初始值。

3. 输出参数

为了把方法的运算结果保存到外部变量，因此需要知道外部变量的引用(地址)。输出参数用于向方法传递外部变量引用(地址)，所以输出参数也是引用参数，与引用参数的差别在于调用方法前无须对变量进行初始化。在方法返回后，传递的变量被认为经过了初始化。值参数、引用参数和输出参数的使用示例如下。

```
using System;
class g{public int a=0;}//类定义
class Class1
{
    public static void F1(ref char i)//引用参数
    { i='b';}
    public static void F2(char i)//值参数，参数类型为值类型
 {    i='d';}
 public static void F3(out char i)//输出参数
 {    i='e';}
    public static void F4(string s)//值参数，参数类型为字符串
 {    s="xyz";}
 public static void F5(g gg)//值参数，参数类型为引用类型
 {    gg.a=20;}
```

```
        public static void F6(ref string s)//引用参数，参数类型为字符串
    {     s="xyz";}
    static void Main(string[] args)
    {
            char str='c';
          string s1="abc";
          F2(str);//值参数，不能修改外部的 a
          Console.WriteLine(str);//因 str 未被修改，显示 c
          F1(ref str);//引用参数,函数修改外部的 a 的值
          Console.WriteLine(str);//str 被修改为 b，显示 b
          Char j;
          F3(out j);//输出参数，结果输出到外部变量 j
          Console.WriteLine(j);//显示 e
          F4(s1);//值参数，参数类型是字符串，s1 为字符串引用变量
          Console.WriteLine(s1);//显示：abc,字符串 s1 不被修改
          g g1=new g();
          F5(g1);//值参数，但实参是一个类引用类型变量
          Console.WriteLine(g1.a.ToString());//显示:20，修改对象数据
          F6(ref s1);//引用参数，参数类型是字符串，s1 为字符串引用变量
          Console.WriteLine(s1);//显示：xyz，字符串 s1 被修改
    }
}
```

4. 数组参数

数组参数使用 params 说明，如果形参表中包含了数组参数，那么它必须是参数表中最后一个参数，数组参数只允许是一维数组。例如，string[]和 string[][]类型都可以作为数组型参数。最后，数组型参数不能再有 ref 和 out 修饰符。示例代码如下。

```
using System;
class Class1
{
    static void F(params int[] args)//数组参数，有 params 说明
    {
        Console.Write("Array contains {0} elements:",args.Length);
        foreach (int i in args)
                Console.Write(" {0}",i);
            Console.WriteLine();
    }
    static void Main(string[] args)
    {
            int[] a = {1,2,3};
        F(a);//实参为数组类引用变量 a
        F(10, 20, 30, 40);//等价于 F(new int[] {60,70,80,90});
        F(new int[] {60,70,80,90});//实参为数组类引用
        F();//等价于 F(new int[] {});
        F(new int[] {});//实参为数组类引用，数组无元素
    }
}
```

运行程序，输出如下。

```
Array contains 3 elements: 1 2 3
Array contains 4 elements: 10 20 30 40
Array contains 4 elements: 60,70,80,90
Array contains 0 elements:
Array contains 0 elements:
```

方法的参数为数组时也可以不使用 params，此种方法可以使用一维或多维数组，示例如下。

```
using System;
class Class1
{
    static void F(int[,] args)//值参数，参数类型为数组类引用变量，无 params 说明
    {
        Console.Write("Array contains {0} elements:",args.Length);
      foreach (int i in args)
            Console.Write(" {0}",i);
      Console.WriteLine();
 }
 static void Main(string[] args)
 {
        int[,] a = {{1,2,3},{4,5,6}};
      F(a);//实参为数组类引用变量 a
      //F(10, 20, 30, 40);//此格式不能使用
      F(new int[,] {{60,70},{80,90}});//实参为数组类引用
      //F();//此格式不能使用
      //F(new int[,] {});//此格式不能使用
 }
}
```

运行程序，输出如下。

```
Array contains 3 elements: 1 2 3 4 5 6
Array contains 4 elements: 60,70,80,90
```

4.10.3　静态方法和实例方法

用修饰符 static 声明的方法为静态方法，不用修饰符 static 声明的方法为实例方法。不管类生成或未生成对象，类的静态方法都可以被使用，使用格式如下。

```
类名. 静态方法名
```

静态方法只能使用该静态方法所在类的静态数据成员和静态方法。这是因为使用静态方法时，该静态方法所在类可能还没有对象，即使有对象，由于用【类名. 静态方法名】方式调用静态方法，静态方法没有 this 指针来存放对象的地址，无法判定应访问哪个对象的数据成员。在类创建对象后，实例方法才能被使用，使用格式如下。

```
对象名. 实例方法名
```

实例方法可以使用该方法所在类的所有静态成员和实例成员。示例代码如下。

```
using System;
public class UseMethod
{
    private static int x=0;//静态字段
    private int y=1;//实例字段
    public static void StaticMethod()//静态方法
    {
        x=10;//正确，静态方法访问静态数据成员
        //y=20;//错误，静态方法不能访问实例数据成员
    }
    public void NoStaticMethod()//实例方法
    {
        x=10;//正确，实例方法访问静态数据成员
        y=20;//正确，实例方法访问实例数据成员
    }
}
public class Class1
{
    public static void Main()
    {
        UseMethod m=new UseMethod();
      UseMethod.StaticMethod();//使用静态方法格式为：类名.静态方法名
      m.NoStaticMethod();//使用实例方法格式为：对象名.实例方法名
    }
}
```

4.10.4　方法的重载

在 C#语言中，如果在同一个类中定义的函数名相同，而参数类型或参数个数不同，认为是不相同的函数，仅返回值不同，不能看作不同函数，这叫做函数的重载。前边 Person 类中定义了多个构造函数就是重载的例子。在 C#语言中，可以使用函数重载特性，对这多个函数定义同样的函数名，但使用不同的参数类型。示例代码如下。

```
using System;
public class UseAbs
{
    public int   abs(int x)//整型数求绝对值
    { return(x<0 ? -x:x);}
    public long abs(long x)//长整型数求绝对值
 {    return(x<0 ? -x:x);}
 public double   abs(double   x)//浮点数求绝对值
 {    return(x<0 ? -x:x);}
}
class Class1
{
    static void Main(string[] args)
    {
        UseAbs m=new UseAbs();
        int x=-10;
        long y=-123;
```

```
            double z=-23.98d;
            x=m.abs(x);
            y=m.abs(y);
            z=m.abs(z);
            Console.WriteLine("x={0},y={1},z={2}",x,y,z);
        }
    }
```

类的对象调用这些同名方法，在编译时，根据调用方法的实参类型决定调用那个同名方法，计算不同类型数据的绝对值。

4.10.5　操作符重载

操作符重载，实际是定义了一个操作符函数，操作符函数声明的格式如下。

```
static public 函数返回类型 operator 重新定义的操作符(形参表)
```

C#语言中有一些操作符是可以重载的。例如：+ - ! ~ ++ -- true false * / % & | ^ << >> == != > < >= <=等。但也有一些操作符是不允许进行重载的。例如：=，&&，||， ?:，new，typeof，sizeof，is 等。

以下例子定义一个复数类，并且希望复数的加减乘除用符号+，-.*，/来表示。

```
using System;
class Complex//复数类定义
{
    private double Real;//复数实部
    private double Imag;//复数虚部
 public Complex(double x,double y)//构造函数
 {
        Real=x;
      Imag=y;
 }
 static public Complex operator - (Complex a)//重载一元操作符负号,注意：一个参数
 {     return (new Complex(-a.Real,-a.Imag));}
 static public Complex operator +(Complex a,Complex b)//重载二元操作符加号
 {     return (new Complex(a.Real+b.Real,a.Imag+b.Imag));}
 public void Display()
 {     Console.WriteLine("{0}+({1})j",Real,Imag);}
}
class Class1
{
    static void Main(string[] args)
    {
        Complex x=new Complex(1.0,2.0);
      Complex y=new Complex(3.0,4.0);
      Complex z=new Complex(5.0,7.0);
      x.Display();//显示:1+(2)j
      y.Display();//显示:3+(4)j
      z.Display();//显示:5+(7)j
      z=-x;//等价于 z=opeator-(x)
```

```
            z.Display();//显示：-1+(-2)j
            z=x+y;//即 z=opeator+(x,y)
            z.Display();//显示：4+(6)j
    }
  }
```

4.10.6　this 关键字

每个类都可以有多个对象。例如，定义 Person 类的两个对象。

```
Person P1=new Person("南宫逸明",20);
Person P2=new Person("暮雨",30);
```

因此 P1，Display()应显示南宫逸明信息，P2，Display()应显示暮雨信息，但无论创建多少个对象，只有一个方法 Display()，该方法是如何知道显示那个对象的信息的呢？C#语言用引用变量 this 记录调用方法 Display()的对象，当某个对象调用方法 Display()时，this 便引用该对象(记录该对象的地址)。因此，不同的对象调用同一方法时，方法便根据 this 所引用的不同对象来确定应该引用哪一个对象的数据成员。this 是类中隐含的引用变量，它是被自动被赋值的，可以使用但不能被修改。例如：P1，Display()，this 引用对象 P1，显示南宫逸明信息。P2，Display()，this 引用对象 P2，显示暮雨信息。

4.10.7　类的多态性

在面向对象的系统中，多态性是一个非常重要的概念。C#支持两种类型的多态性, 第一种是编译时的多态性，一个类的对象调用若干同名方法。系统在编译时，根据调用方法的实参类型及实参的个数决定调用那个同名方法，实现何种操作。编译时的多态性是通过方法重载来实现的。

第二种是运行时的多态性，是在系统运行时，不同对象调用一个名字相同，参数的类型及个数完全一样的方法，会完成不同的操作。C#运行时的多态性通过虚方法实现。在类的方法声明前加上了virtual修饰符，被称之为虚方法，反之为非虚方法。下面的例子说明了虚方法与非虚方法的区别。

```
using System;
class A
{
    public void F()//非虚方法
    { Console.Write("    A.F");}
    public virtual void G()//虚方法
    { Console.Write("    A.G");}
}
class B:A//A 类为 B 类的基类
{
    new public void F()//覆盖基类的同名非虚方法 F()，注意使用 new
    { Console.Write("    B.F");}
    public override void G()//覆盖基类的同名虚方法 G()，注意使用 override
    { Console.Write("    B.G");}
```

```
}
class Test
{
    static void F2(A aA)//注意，参数为 A 类引用变量
    { aA.G();}
    static void Main()
    {
        B b=new B();
        A a1=new A();
        A a2=b;//允许基类引用变量引用派生类对象，a2 引用派生类 B 对象 b
        a1.F();//调用基类 A 的非虚方法 F(),显示 A.F
        a2.F();//F()为非虚方法，调用基类 A 的 F(),显示 A.F
        b.F();//F()为非虚方法，调用派生类 B 的 F(),显示 B.F
        a1.G();//G()为虚方法，因 a1 引用基类 A 对象，调用基类 A 的 G()，显示 A.G
        a2.G();//G()为虚方法，因 a2 引用派生类 B 对象，调用派生类 B 的 G()，显示 B.G
        F2(b);//实参为派生类 B 对象,由于 A aA=b，调用派生类 B 的函数 G(),显示 B.G
        F2(a1);//实参为基类 A 对象,调用 A 类的函数 G(),显示 A.G
    }
}
```

运行程序，输出如下。

```
A.F   A.F   B.F   A.G   B.G   B.G   A.G
```

注意，例子中不同对象调用同名非虚方法 F()和同名虚方法 G()的区别。a2 虽然是基类引用变量，但它引用派生类对象 b。由于 G()是虚方法，因此 a2.G()调用派生类 B 的 G()，显示 G.F。但由于 F()是非虚方法，a2.F()仍然调用基类 A 的 F()，显示 A.F。或者说，如果将基类引用变量引用不同对象，或者是基类对象，或者是派生类对象，用这个基类引用变量分别调用同名虚方法，根据对象不同，会完成不同的操作。而非虚方法则不具备此功能。

方法 F2(A aA)中，参数是 A 类类型，F2(b)中形参和实参的关系是：A aA=b，即基类引用变量 aA 引用派生类对象 b，aA.G()调用派生类 B 的函数 G()，显示 B.G。同理，F2(a1)实参为基类 A 对象,调用 A 类的函数 G(),显示 A.G。

在 Person 类中，公有方法 Display()用来显示个人信息。在派生雇员类 Employee 中，覆盖了基类的公有方法 Display()，以显示雇员新增加的信息。这里希望隐藏这些细节，无论基类还是派生类，都调用同一个显示方法，根据对象不同，自动显示不同的信息。那么可以用虚方法来实现，代码如下。

```
using System;
public class Person
{
 private String name="张三";//类的数据成员声明
 private int age=12;
 protected virtual void Display()//类的虚方法
    {
        Console.WriteLine("姓名:{0},年龄：{1}",name,age);
    }
    public Person(string Name,int Age)//构造函数,函数名和类同名,无返回值
    {
```

```
            name=Name;
            age=Age;
        }
        static public void DisplayData(Person aPerson)//静态方法
        {
            aPerson.Display();//不是静态方法调用实例方法，如写为 Display()错误
        }
    }
    public class Employee:Person//Person 类是基类
    {
        private string department;
        private decimal salary;
        public Employee(string Name,int Age,string D,decimal S):base(Name,Age)
        {
            department=D;
            salary=S;
        }
        protected override void Display()//重载虚方法，注意用 override
        {
            base.Display();//访问基类同名方法
            Console.WriteLine("部门：{0}　薪金：{1} ", department,salary);
        }
    }
    class Class1
    {
        static void Main(string[] args)
        {
            Person OnePerson=new Person("李四",30);
            Person.DisplayData(OnePerson);//显示基类数据
            Employee OneEmployee=new Employee("王五",40,"财务部",2000);
            Person.DisplayData(OneEmployee); //显示派生类数据
        }
    }
```

运行程序，输出结果如下。

```
姓名: 李四,年龄：30
姓名: 王五,年龄：40
部门：财务部　薪金：2000
```

4.11　抽象类和抽象方法

抽象类表示一种抽象的概念，只是希望以它为基类的派生类有共同的函数成员和数据成员。抽象类使用 abstract 修饰符，对抽象类的使用有以下几点规定。

- 抽象类只能作为其他类的基类，它不能直接被实例化。
- 抽象类允许包含抽象成员，虽然这不是必需的。抽象成员用 abstract 修饰符修饰。
- 抽象类不能同时又是密封的。

- 抽象类的基类也可以是抽象类。如果一个非抽象类的基类是抽象类，则该类必须通过覆盖来实现所有继承而来的抽象方法，包括其抽象基类中的抽象方法。如果该抽象基类从其他抽象类派生，还应包括其他抽象类中的所有抽象方法。

示例程序如下。

```
abstract class Figure//抽象类定义
{
    protected double x = 0, y = 0;
    public Figure(double a, double b)
    {
        x = a;
        y = b;
    }
    public abstract void Area();//抽象方法，无实现代码
}
class Square : Figure///类 Square 定义
{
    public Square(double a, double b): base(a, b)
    { }
    public override void Area()//不能使用 new，必须用 override
    { Console.WriteLine("矩形面积是：{0}", x * y); }
}
class Circle : Figure///类 Square 定义
{
    public Circle(double a): base(a, a)
    { }
    public override void Area()
    { Console.WriteLine("园面积是：{0}", 3.14 * x * y); }
}
class Class1
{
    static void Main(string[] args)
    {
        Square s = new Square(20, 30);
        Circle c = new Circle(10);
        s.Area();
        c.Area();
    }
}
```

程序输出结果如下。

```
矩形面积是：600
园面积是：314
```

抽象类 Figure 提供了一个抽象方法 Area()，并没有实现它，类 Square 和 Circle 从抽象类 Figure 中继承方法 Area()，分别具体实现计算矩形和圆的面积。

4.12　密封类和密封方法

有时候，开发者并不希望自己编写的类被继承，或者有的类已经没有再被继承的必要。C#提出了一个密封类(sealed class)的概念，帮助开发人员来解决这一问题。

密封类在声明中使用 sealed 修饰符，这样就可以防止该类被其他类继承。如果试图将一个密封类作为其他类的基类，C#编译器将提示出错。理所当然，密封类不能同时又是抽象类，因为抽象总是希望被继承的。

C#还提出了密封方法(sealed method)的概念。方法使用 sealed 修饰符，称该方法是一个密封方法。在派生类中，不能覆盖基类中的密封方法。

4.13　接口

与类一样，在接口中可以定义一个和多个方法、属性、索引指示器和事件。但与类不同的是，接口中仅仅是它们的声明，并不提供实现。因此，接口是函数成员声明的集合。如果类或结构从一个接口派生，则这个类或结构负责实现该接口中所声明的所有成员。一个接口可以从多个接口继承，而一个类或结构可以实现多个接口。由于 C#语言不支持多继承，因此，如果某个类需要继承多个类的行为时，只能使用多个接口加以说明。

4.13.1　接口声明

接口声明是一种类型声明，它定义了一种新的接口类型。接口声明格式如下。

```
属性　接口修饰符　interface　接口名：基接口{接口体}
```

其中，关键字 interface、接口名和接口体是必需的，其他项是可选的。接口修饰符可以是 new、public、protected、internal 和 private。示例代码如下。

```
public interface IExample
{ //所有接口成员都不能包括实现
  string this[int index] {get;set;}//索引指示器声明
  event EventHandler E;//事件声明
  void F(int value);//方法声明
  string P { get; set;}//属性声明
}
```

声明接口时，需注意以下几点。

- 接口成员只能是方法、属性、索引指示器和事件，不能是常量、域、操作符、构造函数或析构函数，不能包含任何静态成员。
- 接口成员声明不能包含任何修饰符，接口成员默认访问方式是 public。

4.13.2　接口继承

类似于类的继承性，接口也有继承性。派生接口继承了基接口中的函数成员说明。接口允许多继承，一个派生接口可以没有基接口，也可以有多个基接口。在接口声明的冒号后列出被继承的接口名字，多个接口名之间用分号分割。示例代码如下。

```
using System;
interface IControl
{
    void Paint();
}
interface ITextBox:IControl//继承了接口 Icontrol 的方法 Paint()
{
    void SetText(string text);
}
interface IListBox:IControl//继承了接口 Icontrol 的方法 Paint()
{
   void SetItems(string[] items);
}
interface IComboBox:ITextBox,IListBox
{//可以声明新方法}
```

上面的例子中，接口 ITextBox 和 IListBox 都从接口 IControl 中继承，也就继承了接口 IControl 的 Paint 方法。接口 IComboBox 从接口 ITextBox 和 IListBox 中继承，因此它应该继承了接口 ITextBox 的 SetText 方法和 IListBox 的 SetItems 方法，还有 IControl 的 Paint 方法。

4.13.3　类对接口的实现

接口定义不包括函数成员的实现部分，由继承该接口的类或结构来实现这些函数成员。这里主要讲述通过类来实现接口。类实现接口的本质是，用接口规定类应实现那些函数成员。用类来实现接口时，接口的名称必须包含在类声明中的基类列表中。

例如，从 Person 类可以派生出其他类，如工人类、公务员类、医生类等。这些类有一些共有的方法和属性，如工资属性。一般希望所有派生类访问工资属性时用同样变量名。该属性定义在类 Person 中不合适，因为有些人无工资，如小孩。如定义一个类作为基类，包含工资属性，但 C#不支持多继承。可行的办法是使用接口，在接口中声明工资属性。工人类、公务员类、医生类等都必须实现该接口，也就保证了它们访问工资属性时用同样变量名。示例代码如下。

```
using System;
public interface I_Salary//接口
{
     decimal Salary//属性声明
  {
          get;
      set;
```

```
        }
    }
    public class Person
    {…//见前面的 Person 类的定义，这里不重复了。
    }
    public class Employee:Person,I_Salary//Person 类是基类,I_Salary 是接口
    {//不同程序员完成工人类、医生类等，定义工资变量名称可能不同
        private decimal salary;
        public new void Display()
        {
            base.Display();
            Console.WriteLine("薪金：{0} ",salary);
        }
        //工人类、医生类等都要实现属性 Salary，保证使用的工资属性同名
        public decimal Salary
     {
            get
          {     return   salary;}
          set
          {     salary=value;}
     }
    }
    public class Test
    {
        public static void Main()
        {
            Employee S=new Employee();
            S.Name="田七";//修改属性 Name
            S.Age=20;//修改属性 Age
            S.Salary=2000;//修改属性 Salary
            S.Display();
        }
    }
```

如果类实现了某个接口，类也隐式地继承了该接口的所有基接口，不管这些基接口有没有在类声明的基类表中列出。因此，如果类从一个接口派生，则这个类负责实现该接口及该接口的所有基接口中所声明的所有成员。

4.14　代表

在这里要介绍的是 C#的一个引用类型——代表(delegate)，也翻译为委托。它实际上相当于 C 语言的函数指针。与指针不同的是 C#中的代表是类型安全的。代表类声明格式如下。

```
属性集 修饰符 delegate 函数返回类型 定义的代表标识符(函数形参列表);
```

修饰符包括 new、public、protected、internal 和 private。例如，可以声明一个返回类型为 int，无参数的函数的代表 MyDelegate，如下。

```
public delegate int MyDelegate();//只能代表返回类型为 int，无参数的函数
```

声明了代表类 MyDelegate，可以创建代表类 MyDelegate 的对象，用这个对象去代表一个静态方法或非静态的方法，所代表的方法必须为 int 类型，无参数。示例代码如下。

```
using System;
delegate int MyDelegate();//声明一个代表，注意声明的位置
public class MyClass
{
    public int InstanceMethod()//非静态的方法，注意方法为 int 类型，无参数
    {
        Console.WriteLine("调用了非静态的方法。");
        return 0;
    }
    static public int StaticMethod()//静态方法，注意方法为 int 类型，无参数
    {
        Console.WriteLine("调用了静态的方法。");
        return 0;
    }
}
public class Test
{
    static public void Main ()
    {
        MyClass p = new MyClass();
        //用 new 建立代表类 MyDelegate 对象，d 中存储非静态的方法 InstanceMethod 的地址
        MyDelegate d=new MyDelegate(p.InstanceMethod);//参数是被代表的方法
        d();//调用非静态方法
        //用 new 建立代表类 MyDelegate 对象，d 中存储静态的方法 StaticMethod 的地址
        d=new MyDelegate(MyClass.StaticMethod);//参数是被代表的方法
        d();//调用静态方法
    }
}
```

程序的输出结果如下。

```
调用了非静态的方法。
调用了静态的方法。
```

4.15　命名空间

一个应用程序可能包含许多不同的部分，除了自己编制的程序之外，还要使用操作系统或开发环境提供的函数库、类库或组件库，软件开发商处购买的函数库、类库或组件库，开发团队中其他人编制的程序，等等。为了组织这些程序代码，使应用程序可以方便地使用这些程序代码，C#语言提出了命名空间的概念。命名空间是函数、类或组件的容器，把它们按类别放入不同的命名空间中，命名空间提供了一个逻辑上的层次结构体系，使应用

程序能方便地找到所需代码。

4.15.1　命名空间的声明

用关键字 namespace 声明一个命名空间，命名空间的声明要么是源文件 using 语句后的第一条语句，要么作为成员出现在其他命名空间的声明之中。也就是说，在一个命名空间内部还可以定义命名空间成员。全局命名空间应是源文件 using 语句后的第一条语句。在同一命名空间中，不允许出现同名命名空间成员或同名的类。在声明时不允许使用任何访问修饰符，命名空间隐式地使用 public 修饰符。示例代码如下。

```
using System;
namespace N1//N1 为全局命名空间的名称，应是 using 语句后的第一条语句
{
    namespace N2//命名空间 N1 的成员 N2
    {
        class A//在 N2 命名空间定义的类不应重名
        {   void f1(){};}
        class B
        {   void f2(){};}
    }
}
```

也可以采用非嵌套的语法来实现以上命名空间。

```
namespace N1.N2//类 A、B 在命名空间 N1.N2 中
{
    class A
     {      void f1(){};}
    class B
    { void f2(){};}
}
```

也可以采用如下格式。

```
namespace N1.N2//类 A 在命名空间 N1.N2 中
{
    class A
    { void f1(){};}
}
namespace N1.N2//类 B 在命名空间 N1.N2 中
{
    class B
    { void f2(){};}
}
```

4.15.2　命名空间的使用

如在程序中，需引用其他命名空间的类或函数等，可以使用语句 using。例如，需使用上节定义的方法 f1()和 f2()，可以采用如下代码。

```
using N1.N2;
class WelcomeApp
{
    A a=new A();
    a.f1();
}
```

using N1.N2 实际上是告诉应用程序到哪里可以找到类 A。

4.16　本章小结

C#是.NET 中使用得最为广泛的语言。本章简单介绍了 C#语言的基础知识，使得开发人员能够具备 C#语言基础，以便更好地使用 C#语言开发 ASP.NET 应用程序。

本章主要介绍了 C#语言中的基础知识，内容包括类与对象的概念、数据类型及其转换、运算符、流程控制语句、类的继承、类的成员、类的字段和属性、类的方法及参数、抽象类和抽象方法、密封类和密封方法、接口、代表、命名空间等概念及使用方法。

学习完本章内容后，读者应能够使用 C#语言进行开发，以便之后使用 C#更好地进行 ASP.NET 网站开发。

4.17　思考和练习

1. 一列数的规则如下：1、1、2、3、5、8、13、21、34……求第 30 位数是多少， 用递归算法实现。

2. 请编程实现一个冒泡排序算法。

3. CTS、CLS、CLR 分别作何解释？

4. 什么是装箱和拆箱？

5. 数组有没有 length()这个方法？String 有没有 length()这个方法？

6. 谈谈 final，finally，finalize 的区别。

7. 成员变量和成员函数前加 static 的作用？

8. 在 C#中 using 和 new 这两个关键字有什么意义，请写出你所知道的意义？using 指令 和语句 new 创建实例 new 隐藏基类中方法。

第5章　ASP.NET服务器控件

ASP.NET 服务器控件是 ASP.NET 网页上的对象，当客户端浏览器请求服务器端的网页时，这些控件对象将在服务器上运行然后向客户端浏览器呈现 HTML 标记。使用 ASP.NET 服务器控件，可以大幅减少开发 Web 应用程序所需编写的代码量，提高开发效率和 Web 应用程序的性能。

本章首先介绍 ASP.NET 服务器控件的基本概念，然后详细介绍在 ASP.NET 网页中如何添加 ASP.NET 服务器控件，以及这些控件的使用方法。

本章的学习目标：

- 了解 ASP.NET 服务器控件的定义。
- 掌握如何向网页中添加 ASP.NET 服务器控件，如何配置控件的属性、行为和外观以及如何以编程方式使用这些控件。

5.1　ASP.NET 服务器控件概述

在网页上经常看到填写信息用的文本框、单选按钮、复选框、下拉列表等元素，它们都是控件。控件是可重用的组件或对象，有自己的属性和方法，可以响应事件。

ASP.NET 服务器控件是服务器端 ASP.NET 网页上的对象，当用户通过浏览器请求 ASP.NET 网页时，这些控件将运行并把生成的标准的 HTML 文件发送给客户端浏览器来呈现。

网站部署在 Web 服务器上，人们可以通过浏览器来访问这个站点。当客户端请求一个静态的 HTML 页面时，服务器找到对应的文件直接将其发送给用户端浏览器；而在请求 ASP.NET 页面时(扩展名为.aspx 的页面)，服务器将在文件系统中找到并读取对应的页面，然后将页面中的服务器控件转换成浏览器可以读取的 HTML 标记和一些脚本代码，再将转换后的结果页面发送给用户。

在 ASP.NET 页面上，服务器控件表现为一个标记，如<asp:textbox…/>。这些标记不是标准的 HTML 元素，因此如果它们出现在网页上，浏览器将无法理解它们。然而，当从 Web 服务器上请求一个 ASP.NET 页面时，这些标记都将动态地转换为 HTML 元素，因此浏览器只会接收到它能理解的 HTML 内容。

在创建.aspx 页面时，可以将任意的服务器控件放置到页面上，然而请求服务器上该页面的浏览器将只会接收到 HTML 和 JavaScript 脚本代码，如图 5-1 所示。

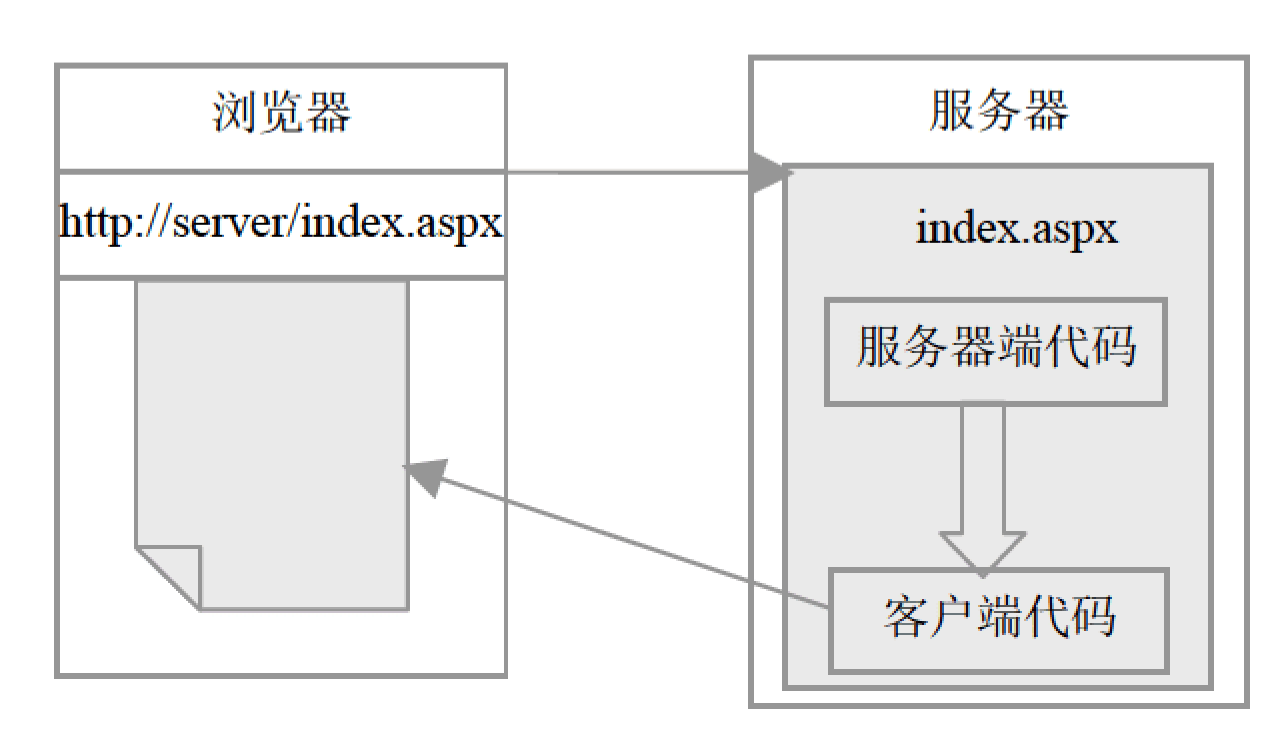


图 5-1　服务器端代码转换为客户端代码

由此可见，Web 浏览器无法理解 ASP.NET。Web 浏览器只能理解 HTML，可能也理解 JavaScript，但它不能直接处理 ASP.NET 代码。服务器读取 ASP.NET 代码并进行处理，然后将所有 ASP.NET 特有的内容转换为 HTML 以及(如果浏览器支持)一些 JavaScript 代码，然后将最新生成的 HTML 发送回浏览器。

ASP.NET 服务器控件主要类别有 HTML 服务器控件、标准服务器控件、验证控件、用户控件、数据控件、导航控件等。

本章主要介绍 HTML 服务器控件、标准服务器控件、验证控件和用户控件。

5.2　HTML 元素和 HTML 服务器控件

本节描述如何使用 VS 工具箱的 HTML 选项卡中的 ASP.NET 服务器控件。

5.2.1　HTML 元素

在 VS 中，从工具箱添加到页面上的 HTML 服务器控件只是已设置了某些属性的 HTML 元素，当然也可通过在【源】视图中输入 HTML 标记创建 HTML 元素。

默认情况下，ASP.NET 文件中的 HTML 元素作为文本进行处理，并且不能在服务器端代码中引用这些元素，只能在客户端通过 javascript 和 vbscript 等脚本语言来控制。

工具箱的 HTML 选项卡上提供了一些基于 HTML INPUT 元素的控件。本节通过下列两个控件介绍 HTML INPUT 元素的使用方法。

(1) Input (Text) 控件，文本框控件：默认情况下是 INPUT type="text"元素。

(2) Input (Button)控件，按钮控件：默认情况下是 INPUT type="button"元素。

Input(Text)元素创建允许用户在其中输入文本或密码的单行文本框，其主要属性有以下几点。

- Type=text/password，文本框的类型。
- MaxLength，文本框中最大的输入字节。
- Size，设定文本框的宽度。
- Value，设定文本框的值。

Input(Button)元素的主要功能是创建一个用来触发事件处理程序的按钮，通过使用 onClick 属性来表明单击按钮可以触发的处理方法。其主要属性有以下几点。

- ID，此控件的编程名称。
- value，设置按钮中显示的文本。

【例 5-1】演示如何使用 Input (Text)元素和 Input (Button)元素。

(1) 运行 VS，新建名为 WebSite5 的 ASP.NET 网站。

(2) 新建页面 example5-1.aspx，切换到【设计】视图，从【工具箱】的 HTML 组中拖动 3 个 Input(Text)控件和 1 个 Input(Button)控件到页面，页面设计效果如图 5-2 所示，再切换到【源】视图可看到如下所示的代码。

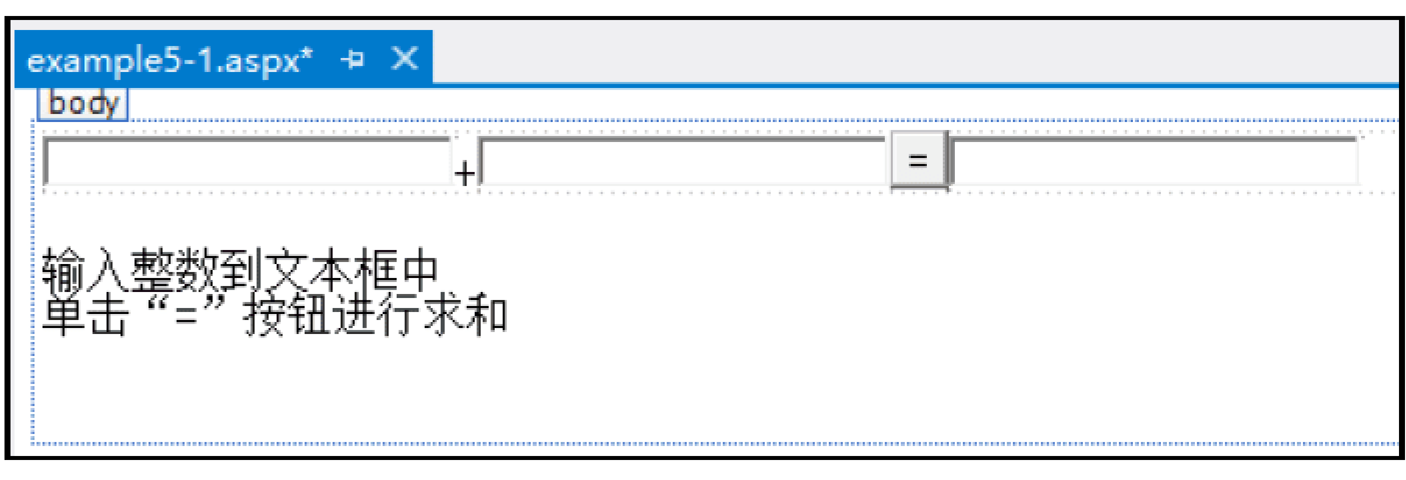


图 5-2　HTML image 控件

```
<input id="Text1" type="text" />+
<input id="Text2" type="text" />
<input id="Button1" type="button" value="=" />
<input id="Text3" type="text" /></div>
```

为文本框 Text1 和 Text2 分别添加属性 maxlength ="3"以控制输入字符的个数不超过 3 个。给按钮 Button1 控件添加属性 onclick="AddButton_Click();"。

(3) 切换到【源】视图，添加如下客户端脚本到<body>标记前。

```
<script type ="text/javascript">
      function AddButton_Click( )
      {
            alert('You clicked me.');
            var a= document.getElementById("Text1").value;
            var b=   document.getElementById("Text2").value;
            // alert(a);    alert(b);
            var answer=parseInt(a)+parseInt(b);
            document.getElementById("Text3").value=answer ;
            //alert(answer);
      }
</script>
```

(4) 按 Ctrl+F5，观看运行效果。单击=按钮时，将调用客户端脚本 AddButton_Click()。

5.2.2　HTML 服务器控件

在 VS 中，从工具箱添加到 ASP.NET 页面上的 HTML 服务器控件只是已设置了某些属性的 HTML 元素。

默认情况下，这些添加到 ASP.NET 文件中的 HTML 元素被视为传递给浏览器的标记，作为文本进行处理，不能在服务器端的代码中引用这些元素。若要使这些元素能以编程方式进行访问，可以通过添加 runat="server"属性表明应将 HTML 元素作为服务器控件进行处理。还可设置 HTML 元素的 id 属性，这样就可使用基于服务器的代码对其进行编程引用了。

添加了属性 runat="server"的 HTML 元素就转换为 HTML 服务器控件了。移除控件标记中的 runat="server"属性，HTML 服务器控件就转换为 HTML 元素了。

【例 5-2】演示如何使用工具箱的 Input(Button)服务器控件、Input(Text)服务器控件。

(1) 打开【例 5-1】所建的 WebSite5 网站。

(2) 新建名为 example5-2.aspx 的 Web 页面文件。从【工具箱】的 HTML 组中拖动 3 个 Input(Text)控件和 1 个 Input(Button)控件到页面，如图 5-3 所示。

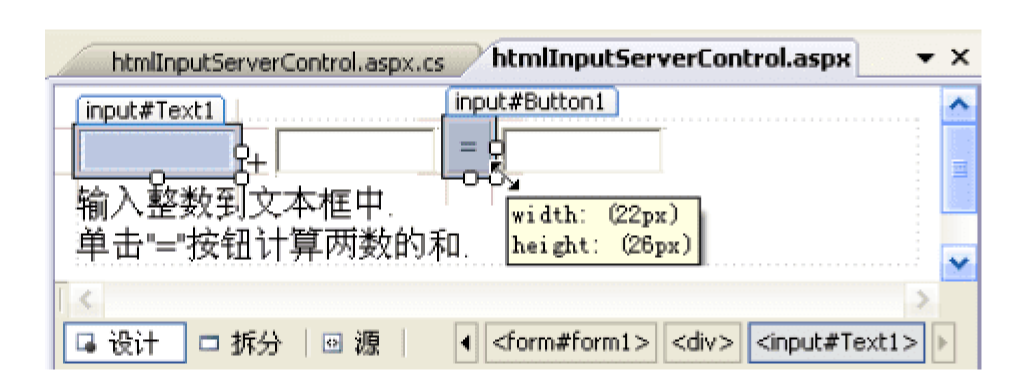


图 5-3　HTML input 控件设计视图

为文本框 Text1 和 Text2 分别添加属性 maxlength ="3"以控制输入字符的个数不超过 3 个。给按钮 Button1 控件添加属性 onserverclick="AddButton_Click"，重置值 value="="。

给这 3 个文本框控件和这 1 个按钮控件都添加属性 runat="server"，以便将这些控件转换为 HTML 服务器控件。

切换到【源】视图，可看到如下所示的代码。

```
<input id="Text1" type="Text" size="2" maxlength="3" runat="server" /> +
<input id="Text2" type="Text" size="2" maxlength="3" runat="server" />
<input id="Button1" type="button" value="=" onserverclick="AddButton_Click" runat="server"/>
<input id="Text3" type="Text" size="2" maxlength="3" runat="server" />
```

(3) 切换到 example5-2.aspx 的【源】视图，将下列服务器端脚本加到<body>标记前。

```
<script runat="server">
    protected void AddButton_Click(Object sender, EventArgs e)
    {
        int Answer;
        Answer = Convert.ToInt32(Text1.Value) + Convert.ToInt32(Text2.Value);
        Text3.Value = Answer.ToString();
    }
</script>
```

(4) 按 Ctrl+F5 组合键，在 IE 浏览器中观看运行效果。单击=按钮时，将调用服务器端脚本 AddButton_Click()。

HTML 控件由 HTML 元素衍生而来，由于 HTML 元素的属性只能静态地设置，一般在程序执行过程中不能被修改，很不灵活。为了弥补这一不足。ASP.NET 特地提供 HTML 控件，这种控件允许在程序中设置其属性，也允许在程序的执行过程中动态地读取及修改其属性，从而可以产生动态的网页。

【例 5-1】中的 HTML Input(Button)元素和【例 5-2】HTML 服务器控件的比较，代码

如下。

```
<input id="Button1" type="button" onclick="AddButton_Click();" value="=" />
<input id="Button1" type="button onserverclick="AddButton_Click" runat="server" " value="="/>
```

比较这两个语句，可以看出声明一个 HTML 服务器按钮控件和声明一个 HTML 元素有以下几点不同。

- 服务器按钮控件用 onserverclick 属性代替了 html 按钮的 onclick 属性。
- 服务器按钮控件多用了一个属性 runat＝"server"。

说明：

- ID 属性是标识服务器控件的唯一标志，通过它可以像引用一个对象一样来直接在服务器端引用服务器控件。
- 用 OnServerClick 属性代替 OnClick 属性是为了表明响应按钮的单击事件是在服务器端进行处理，而不是客户端。
- 多出的 runat＝"server"属性是说明该控件为服务器控件，这是区别 HTML 服务器控件和 html 元素的唯一方法。

5.3　标准服务器控件

ASP.NET 包含大量可在 ASP.NET 网页上使用的标准服务器控件。本节重点从 ASP.NET 服务器控件的工作原理和过程入手，介绍控件的常用功能和用法。

5.3.1　Button 控件

可使用 ASP.NET Button 服务器控件为用户提供向服务器发送网页的能力。该控件会在服务器代码中触发一个事件，可以处理该事件来响应回发。

ASP.NET 中有 3 种按钮控件，每种按钮控件在网页上显示的方式都不同，如表 5-1 所示。

表 5-1 按钮控件的比较

控　　件	说　　明
Button	显示一个标准命令按钮，该按钮呈现为一个 HTML input 元素
LinkButton	呈现为页面中的一个超链接。但是，它包含使窗体被发回服务器的客户端脚本(可以使用 HyperLink 服务器控件创建真实的超链接)
ImageButton	将图形呈现为按钮。这对于提供丰富的按钮外观非常有用。ImageButton 控件还提供有关图形内已单击位置的坐标信息

(1) 按钮事件

当用户单击任何 Button(按钮)服务器控件时，会将该页发送到服务器。这使得在基于

服务器的代码中，网页被处理，任何挂起的事件被触发。这些按钮还可触发它们自己的 Click 事件，可以为这些事件编写事件处理程序。

(2) 按钮回发行为

当用户单击按钮控件时，该页回发到服务器。默认情况下，该页回发到其本身，在这里重新生成相同的页面并处理该页上控件的事件处理程序。

可以配置按钮以将当前页面回发到另一页面。这对于创建多页窗体可能非常有用。

默认情况下，Button 控件使用 HTML POST 操作提交页面。LinkButton 和 ImageButton 控件不能直接支持 HTML POST 操作。因此，使用这些按钮时，它们将客户端脚本添加到页面以允许控件以编程方式提交页面。因此 LinkButton 和 ImageButton 控件要求在浏览器上启用客户端脚本。

在某些情况下，可能希望 Button 控件也使用客户端脚本执行回发。这在希望以编程方式操作回发(如将回发附加到页面上的其他元素)时非常有用。可以将 Button 控件的 UseSubmitBehavior 属性设置为 true，以使 Button 控件使用基于客户端脚本的回发。

(3) 处理 Button 控件的客户端事件

Button 控件既可以引发服务器事件，也可以引发客户端事件。服务器事件在回发后发生，且这些事件在为页面编写的服务器端代码中处理。客户端事件在客户端脚本(通常为 ECMAScript(JavaScript))中处理，并在提交页面前引发。通过向 ASP.NET 按钮控件添加客户端事件，可以执行一些任务(如在提交页之前显示确认对话框以及可能取消提交)。

【例 5-3】演示 Button 控件、LinkButton 控件、ImageButton 控件的使用。

① 启动 VS，然后打开 ASP.NET 网站 WebSite5。

② 新建页面 example5-3.aspx、music.aspx 和 dog.aspx。在网站根目录下新建名为 Images 的文件夹，将选中的图片添加到 Images 文件夹中。

③ 切换到 example5-3-1.aspx 的【设计】视图，在【工具箱】中双击相应的控件，添加 1 个 HyperLink 控件、1 个 LinkButton 控件、1 个 Button 控件、1 个 ImageButton 控件和 1 个 Label 控件，如图 5-4 所示。

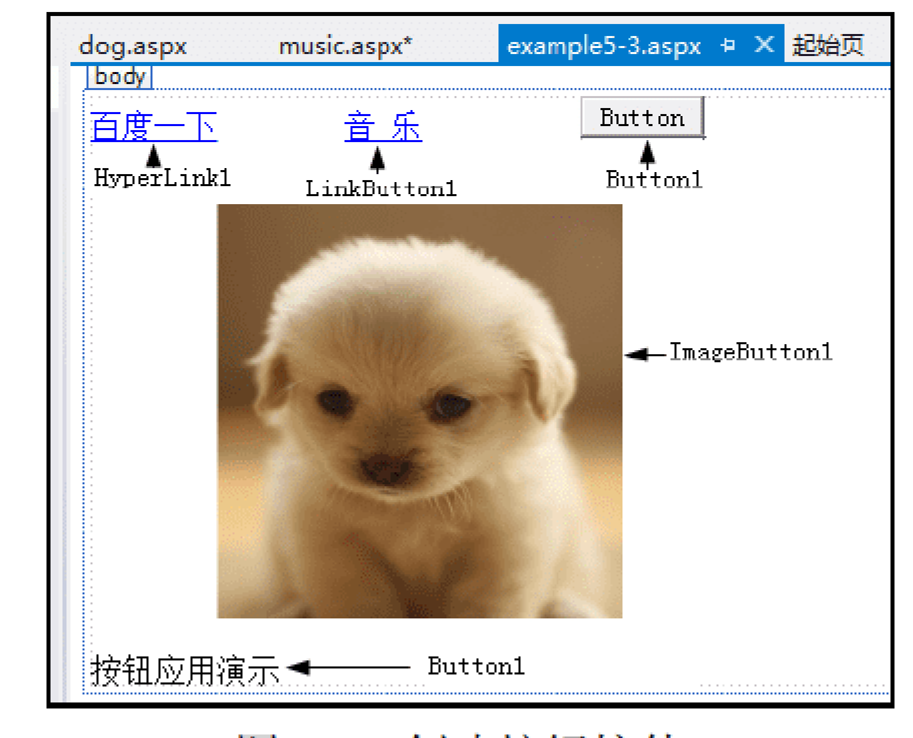


图 5-4　创建按钮控件

④ 在【设计】视图中，单击 HyperLink1 控件，在其属性窗口设置 Text 属性值为 Baidu，NavigateUrl 属性值分别为 http://www.baidu.com。

同样，设置 LinkButton1 控件的 Text 属性值为【音乐】，PostBackUrl 属性值为~/music.aspx；ImageButton1 控件的 ImageUrl 属性值为~/Images/dog.jpg，PostBackUrl 属性值为~/dog.aspx，如下标记代码所示。

```
<div>
<asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="http://www.baidu.com/">百度一下
</asp:HyperLink>
```

```
<asp:LinkButton ID="LinkButton1" runat="server" PostBackUrl="~/music.aspx"
onclick="LinkButton1_Click">音 乐</asp:LinkButton>
    <asp:Button ID="Button1" runat="server" Text="Button" onclick="Button1_Click" />
    <asp:ImageButton ID="ImageButton1" runat="server" Height="209px" ImageUrl="~/Images/dog.jpg"
PostBackUrl="~/dog.aspx" Width="205px" onclick="ImageButton1_Click" AlternateText="this is a
ImageButton."/>
    <asp:Label ID="Label1" runat="server" Text="按钮应用演示"></asp:Label>
    </div>
```

⑤ 在【设计】视图中，双击 Button1 控件，为 Button1 的 Click 事件添加如下所示的处理程序。

```
protected void Button1_Click(object sender, EventArgs e)
{
    Label1.Text = "您单击了 Button 按钮！";
}
```

⑥ 按 Ctrl+F5 组合键，观看显示效果。

5.3.2 TextBox(文本框)控件

TextBox 服务器控件允许用户向页面输入文本。默认情况下，该控件的 TextMode 属性设置为 TextBoxMode.SingleLine，即单行文本框。若将 TextMode 属性设置为 TextBoxMode.MultiLine，可以显示为多行文本框(该文本框将作为 HTML 中的 textarea 元素呈现)。也可以将 TextMode 属性更改为 TextBoxMode.Password，即密码文本框。通过使用 Text 属性可以获得 TextBox 控件中显示的文本。

【例 5-4】演示文本框控件 TextBox、标签控件 Label、按钮控件 Button 的使用。

(1) 启动 VS，打开网站 WebSite5。

(2) 新建网页 example5-4.aspx。

(3) 切换到如图 5-5 所示的【设计】视图，添加名为 Label 控件 Label1、Label2，TextBox 控件 TextBox1、TextBox2 的 TextBox，Button 控件 Button1。

(4) 单击 Label1 控件，在其属性窗口设置 Text 属性值为“初始内容：”；同样，设置 Label2 控件的 Text 属性值为“克隆内容：”，Button1 控件的 Text 属性值为“克隆”。如图 5-4 所示。

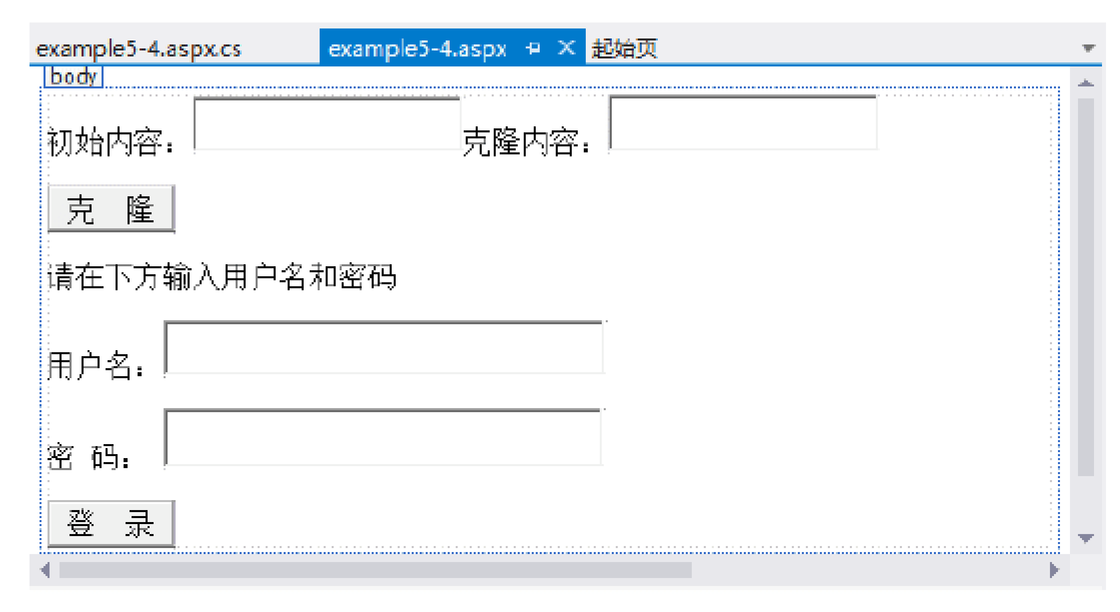


图 5-5　创建控件

(5) 在【设计】视图中，双击 Button1 控件，切换到 example5-4.aspx.cs 视图，为 Button1 控件的 Click 事件添加如下所示的处理程序。

```
protected void Button1_Click(object sender, System.EventArgs e)
{
    //读取 TextBox1 控件的 Text 属性的值，并使用该值来设置 TextBox2 控件的 Text 属性值。
```

```
        TextBox2.Text = Server.HTMLEncode(TextBox1.Text);
    }
```

(6) 按 Ctrl+F5 组合键，观看显示效果。

(7) 在【设计】视图中，分别单击 TextBox1、TextBox2 控件，在其属性窗口设置 TextMode(文本框的行为模式)属性值为 MultiLine。按 Ctrl+F5 组合键，观看显示效果。

(8) 按图 5-5 所示，添加 3 个 Label 控件，2 个 TextBox 控件，1 个 Button 控件，在属性窗口分别如图 5-5 所示设置 Text；设置 TextBox4 控件的 TextMode 属性值为 Password。

(9) 在【设计】视图中，双击【登录】按钮控件，切换到 example5-4.aspx.cs 视图，为 Button2 控件的 Click 事件的处理方法添加如下所示的代码。

```
protected void Button2_Click(object sender, EventArgs e)
{
    if (TextBox3.Text ==" " || TextBox4.Text == "")
    {
        string scriptString = "alert('信息不全，请重新填写。'); ";
        Page.ClientScript.RegisterClientScriptBlock(this.GetType(),"warning", scriptString, true);
    }
    else
    {
        Response.Redirect(string.Format("Content5-4.aspx?user={0}&password={1}",
        TextBox3.Text, TextBox4.Text));
    }
```

在这段代码中，当单击【登录】按钮后，首先检查【用户名】文本框和【密码】文本框中的内容是否为空，若为空，则弹出对话框，提示用户重新输入信息；若不空，则将控件中的数据作为参数传给 Content5-4.aspx 页面。

(10) 在 Content5-4.aspx 页面的【设计】视图下，双击 Web 窗体，在 Page_Load 事件中添加以下代码。

```
protected void Page_Load(object sender, EventArgs e)
        {
            string scriptContent = Request.QueryString["user"].ToString() + "，欢迎您！请记住您的密
码：" + Request.QueryString["password"].ToString();
            Response.Write(scriptContent);
            string scriptString = "alert('"+scriptContent+"');";
            Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "success", scriptString, true);
        }
```

(11) 按 Ctrl+F5 组合键，观看显示效果。

5.3.3　ListBox 控件和 DropDownList 控件

ListBox 服务器控件允许用户从预定义的列表中选择一项或多项。

DropDownList 服务器控件允许用户从预定义的下拉列表中选择单项，它与 ListBox 服务器控件的不同之处在于，其选项列表在用户单击下拉按钮之前一直保持隐藏状态。此外，DropDownList 控件不支持多重选择模式。

DropDownList 控件的某个选项被选中时，该控件将触发 SelectedIndexChanged 事件。默认情况下，此事件不会导致向服务器发送页，但可通过将 AutoPostBack 属性设置为 true，强制该控件立即发送。

如果将 ListBox 控件的 SelectionMode 属性设置为 Multiple，将允许用户多选，用户可以在按住 Ctrl 或 Shift 键的同时，单击选择多个选项。

【例 5-5】演示 ListBox 控件和 DropDownList 控件的使用。

(1) 启动 VS，打开网站 WebSite5，新建名为 example5-5 的 ASP.NET 网站。

(2) 切换到【设计】视图，从【工具箱】中拖动 1 个 DropDownList 控件、1 个 ListBox 控件和 1 个 Label、1 个 Button 控件到页面，如图 5-6 所示。

(3) 选择 DropDownList1 控件，单击右上角的【智能标记】，选中【启用 AutoPostBack】，再单击【编辑项】，弹出如图 5-7 所示的对话框，单击【添加】按钮，添加 4 个成员并设置相应的 Text 属性值。

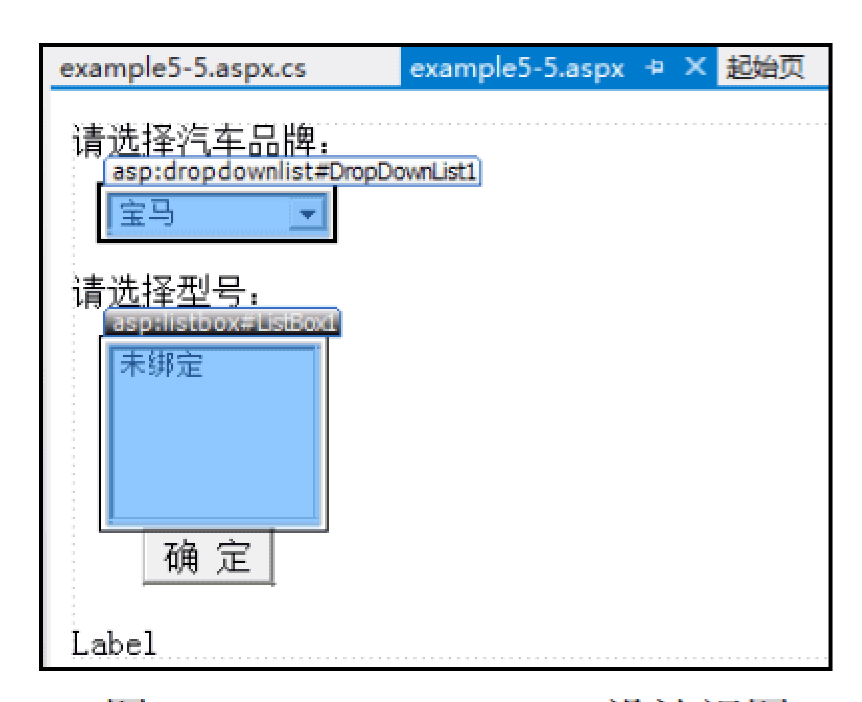



图 5-6　example5-5.aspx 设计视图

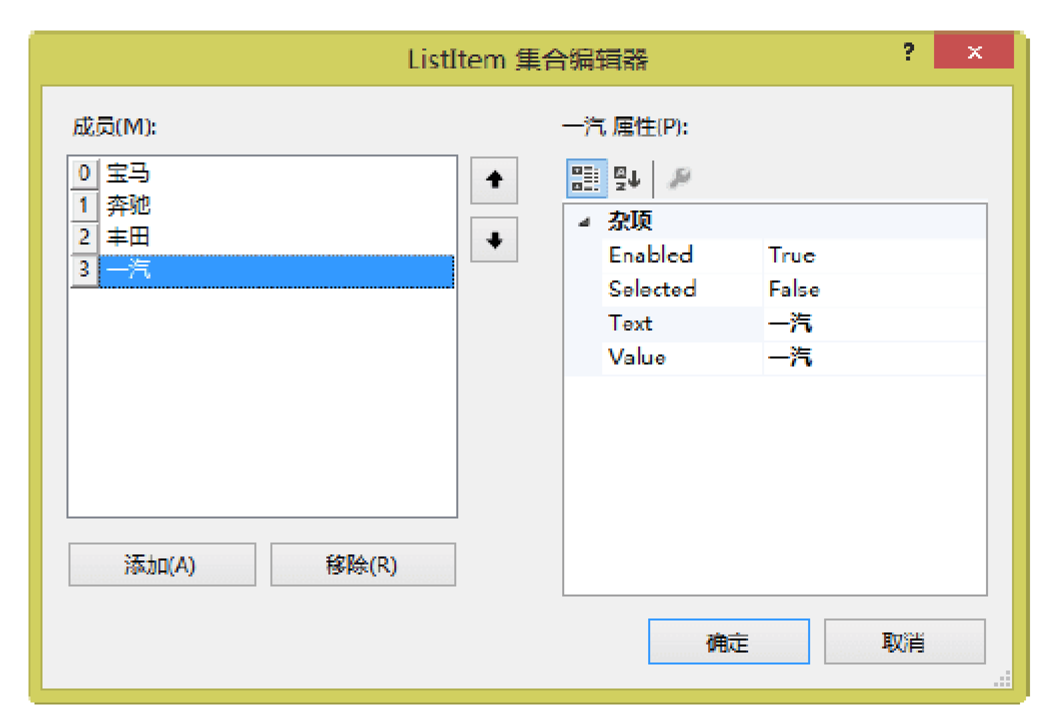


图 5-7　ListItem 集合编辑器

(4) 双击 DropDownList1 控件，添加其 SelectedIndexChanged 事件，代码如下。

```
protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
        {
            switch (DropDownList1.SelectedItem.Text.Trim())
            {
                case "宝马":
                    ListBox1.Items.Clear();
                    ListBox1.Items.Add("轿车-宝马 3 系");
                    ListBox1.Items.Add("SUV-宝马 X5");
                    ListBox1.Items.Add("跑车-宝马 i8 ");
                    break;
                case "奔驰":
                    ListBox1.Items.Clear();
                    ListBox1.Items.Add("轿车-奔驰 C 级");
                    ListBox1.Items.Add("SUV-奔驰 GLK 级");
                    ListBox1.Items.Add("MPV-奔驰 R 级");
                    break;
                case "丰田":
                    ListBox1.Items.Clear();
                    ListBox1.Items.Add("轿车-卡罗拉");
                    ListBox1.Items.Add("MPV-逸致");
```

```
                    ListBox1.Items.Add("SUV-汉兰达");
                    break;
                case "一汽":
                    ListBox1.Items.Clear();
                    ListBox1.Items.Add("轿车-夏利 N7");
                    ListBox1.Items.Add("SUV-骏派 D60");
                    ListBox1.Items.Add("MPV-森雅 M80");
                    break;
            }
        }
```

(5) 双击 Button1 控件，添加其 Click 事件，代码如下。

```
protected void Button1_Click(object sender, EventArgs e)
        {
            string msg = "您选择的爱车是：";
            foreach (ListItem li in ListBox1.Items)
            {
                if (li.Selected == true)
                {
                    msg += "<br />" + li.Text;
                }
            }
            Label1.Text = msg;
        }
```

(6) 选中 ListBox1 控件，在【属性】窗口设置属性 SelectionMode 的值为 Multiple。

(7) 按 Ctrl+F5 组合键，分别选择不同的选项，观看显示效果。

5.3.4　CheckBox(复选框)和 CheckBoxList(复选框列表)控件

CheckBox 控件和 CheckBoxList 控件分别用于向用户提供选项和选项列表。CheckBox 控件适合用在选项不多且比较固定的情况，当选项较多或需在运行时动态决定有哪些选项时，使用 CheckBoxList 控件比较方便。

【例 5-6】演示复选框控件 CheckBox 和复选框列表控件 CheckBoxlist 的使用。

(1) 启动 VS，打开网站 WebSite5，新建网页 example5-6.aspx。

(2) 切换到 example5-6.aspx 页的【设计】视图。

(3) 输入静态文本“CheckBoxList 控件演示程序”，在【格式】工具栏中使用【块格式】下拉列表将该文本的格式设置为“标题 1”。

(4) 添加 CheckBoxList1、Button1、CheckBox1、CheckBox2 和 Label1 这 5 个控件，如图 5-8 所示。

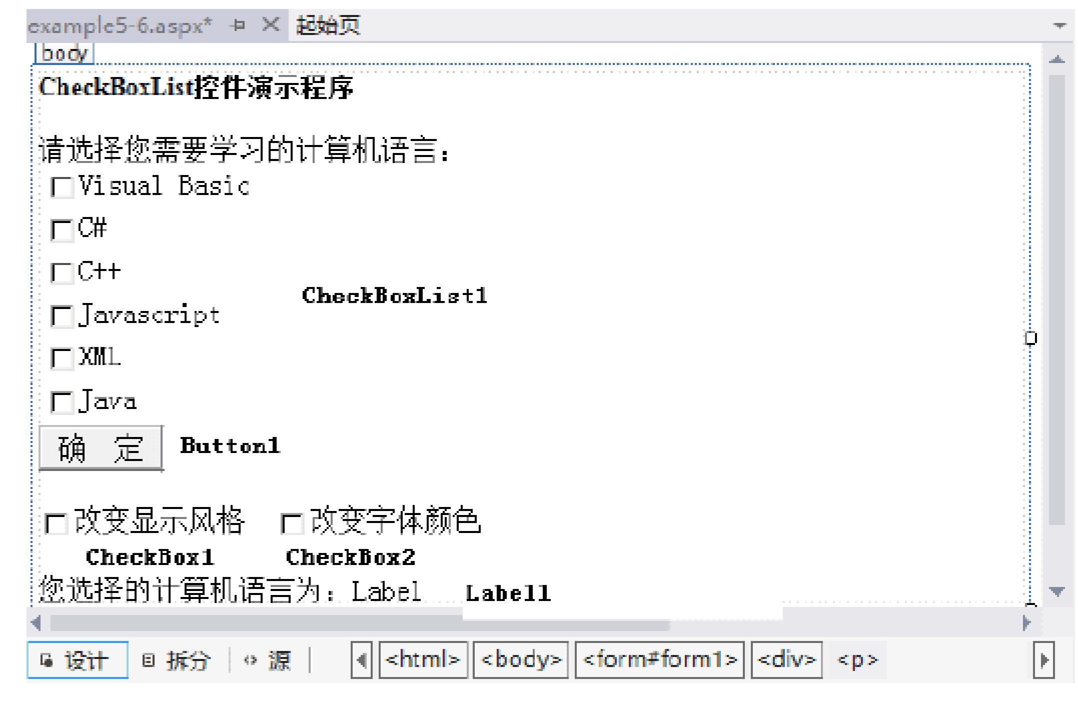


图 5-8　添加控件

(5) 在【设计】视图中，单击 CheckBoxList1 控件右上角的【智能标

记】，切换到【CheckBoxList 任务】对话框，不选择【启用 AutoPostBack】，单击【编辑项】，出现【ListItem 集合编辑器】，单击【添加】按钮，添加成员 ListItem0~ListItem5，分别将其属性 Text 值设置为 Visual Basic、C#、C++、Jscript、XML、Java。其 HTML 代码如下。

```
<asp:CheckBoxList ID="CheckBoxList1" runat="server" Height="16px" Width="221px">
                <asp:ListItem>Visual Basic</asp:ListItem>
                <asp:ListItem>C#</asp:ListItem>
                <asp:ListItem>C++</asp:ListItem>
                <asp:ListItem>Javascript</asp:ListItem>
                <asp:ListItem>XML</asp:ListItem>
                <asp:ListItem>Java</asp:ListItem>
 </asp:CheckBoxList>
```

(6) 设置 Button1 控件的 Text 属性值为【确定】，CheckBox1 控件的 Text 属性值为【改变显示风格】，CheckBox2 控件的 Text 属性值为【改变字体颜色】，如图 5-8 所示。

(7) 在【设计】视图中，分别双击 CheckBox1、CheckBox2 和 Button1 控件，切换到代码视图，添加如下所示的代码。

```
protected void Button1_Click(object sender, EventArgs e)
{
    string str = "选择结果：";
    Label1.Text = "";
    for (int i = 0; i < CheckBoxList1.Items.Count; i++)
    {
        if (CheckBoxList1.Items[i].Selected)
        {
            //Label1.Text += CheckBoxList1.Items[i].Text + " <br/> ";
            str += CheckBoxList1.Items[i].Text + "、";
        }
    }
    if (str.EndsWith("、") == true) str = str.Substring(0, str.Length - 1);
    Label1.Text = str;
    if (str == "选择结果：")
    {
        string scriptString = "alert('请作出选择！');";
        Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "warning!",
             scriptString, true);
    }
    else
    {
        Label1.Visible = true;
        Label1.Text = str;
    }
}
protected void CheckBox1_CheckedChanged(object sender, EventArgs e)
{
    this.CheckBoxList1.BackColor =
        CheckBox1.Checked ? System.Drawing.Color.Beige : System.Drawing.Color.Azure;
    CheckBoxList1.RepeatDirection =
```

```
            CheckBox1.Checked ? RepeatDirection.Horizontal : RepeatDirection.Vertical;
}
protected void CheckBox2_CheckedChanged(object sender, EventArgs e)
{
    if (CheckBox2.Checked)
    {
    this.CheckBoxList1.ForeColor = System.Drawing.Color.Red;
        Label1.ForeColor = System.Drawing.Color.Red;
    }
    else
    {
        this.CheckBoxList1.ForeColor = System.Drawing.Color.Black;
        Label1.ForeColor = System.Drawing.Color.Black;
    }
}
```

(8) 按 Ctrl+F5 组合键，观看显示效果。

5.3.5　Image 和 ImageMap 控件

ASP.NET 4.5 包含两个图形控件，一个是 Image 控件，一个是 ImageMap 控件。

(1) Image 服务器控件允许用户在 ASP.NET 网页上显示图像，并用自己的代码管理这些图像。可以在设计时或运行时以编程方式为 Image 对象指定图形文件。还可以将控件的 ImageUrl 属性绑定到一个数据源，以根据数据库信息显示图形。

与大多数其他服务器控件不同，Image 控件不支持任何事件。例如，Image 控件不响应鼠标单击事件。实际上，可以通过使用 ImageMap 或 ImageButton 控件来创建交互式图像。

(2) Imagemap 说白了就是一种图形，包括许多不同部分，将鼠标指针指在图形的各个部分，单击，就可以进入另一个超级链接的页面。

Imagemap 不一定要真的是 map，可以是任何图形。例如，一张脸，单击眼睛就出现关于眼睛的页面，单击鼻子就出现鼻子的页面，也是一种 ImageMap。

ImageMap 控件由两个元素组成。第一个是图像，它可以是任何标准 Web 图形格式的图形，如.gif、.jpg 或.png 文件。第二个元素是 HotSpot(作用点)控件的集合。每个作用点控件都是一个类型为 CircleHotSpot、RectangleHotSpot 或 PolygonHotSpot 的不同项。对于每个作用点控件，都要定义用于指定该作用点的位置和大小的坐标。例如，如果创建一个 CircleHotSpot 控件，则需要定义圆心的 x 和 y 坐标以及圆的半径。

响应用户单击：每一个作用点都可以是一个单独的超链接或回发事件。可以指定用户单击作用点时发生的事件，可以将每个作用点配置为可以转到为该作用点提供的 URL 的超链接。或者，也可以将控件配置为在用户单击某个作用点时执行回发，并可为每个作用点提供一个唯一值。回发会触发 ImageMap 控件的 Click 事件。在事件处理程序中，可以读取分配给每个作用点的唯一值。

【例 5-7】演示 Image 控件和 ImageMap 控件的使用。

① 启动 VS，打开站点 WebSite5，新建一个网页 example5-7.aspx。

使用之前创建好的 Images 文件夹，将准备好的图片复制到 Images 文件夹中。

② 切换到 example5-7.aspx 页的【设计】视图。

③ 输入静态文本“ImageMap 控件导航示例”，在【格式】工具栏中使用【块格式】下拉列表将该文本的格式设置为“标题 1”；然后在下方输入“单击某个行星图片可看其更大的图片”。

④ 如图 5-9 所示，添加 ImageMap1、Image1 和 Image2 这 3 个控件。

⑤ 在【设计】视图中，单击 ImageMap 控件，在其【属性】窗口，设置属性值为：id="solarsys"、imageurl="~ /images/xingxing.jpg"、alternatetext="Solar System(太阳系) "，再单击 HotSpots(作用点)属性框来显示【HotSpots 集合编辑器】对话框，单击【添加】按钮旁边的箭头按钮来选择要创建的作用点类型(CircleHotSpot、RectangleHotSpot、PolygonHotSpot)，如图 5-10 所示，添加 2 个成员，并根据如下所示的语法设置其相应属性。

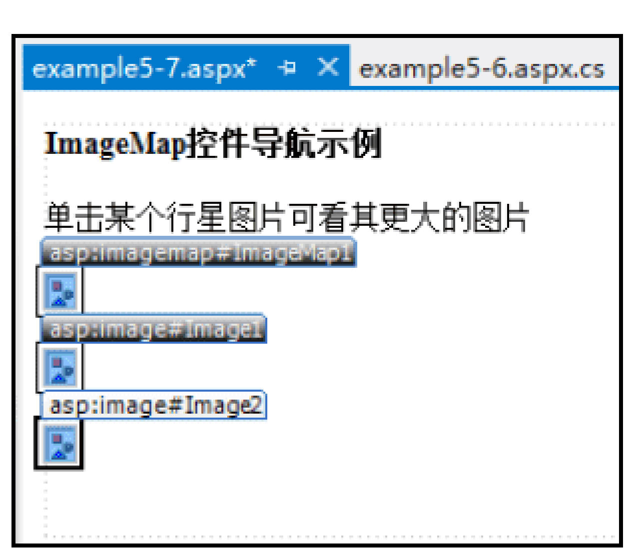




图 5-9　添加 ImageMap 控件和 Image 控件

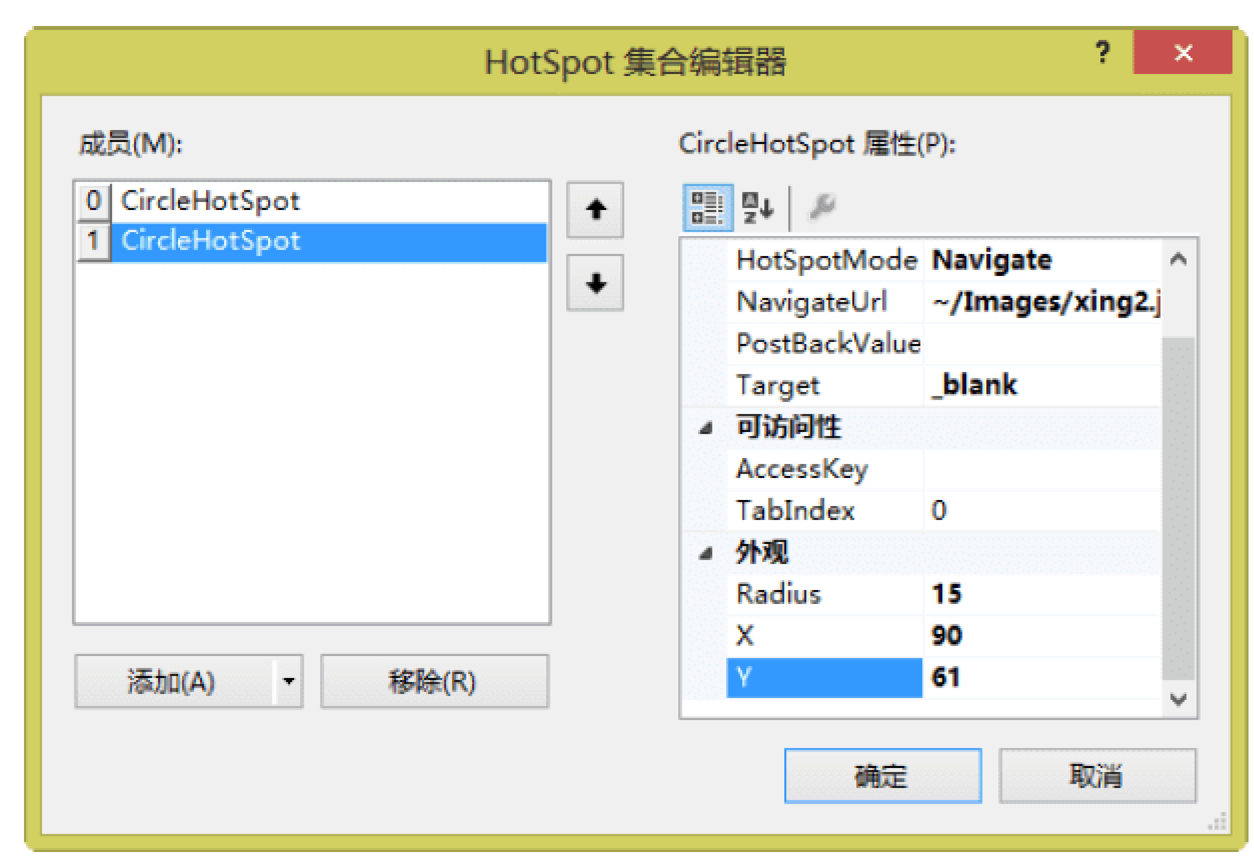


图 5-10　为 ImageMap 控件添加 HotSpot 热区

ImageMap 控件的 HTML 语法如下。

```
    <asp:ImageMap ID="solarsys" runat="server" AlternateText="Solar System(太阳系)"
Height="212px" HotSpotMode="Navigate" ImageUrl="~/images/xingxing.jpg" Width="440px">
    <asp:CircleHotSpot HotSpotMode="Navigate" NavigateUrl="~/Images/xing1.jpg" Radius="30"
Target="_blank" X="47" Y="162" />
    <asp:CircleHotSpot
HotSpotMode="Navigate"
NavigateUrl="~/Images/xing2.jpg" Radius="30"
Target="_blank" X="208" Y="126" />
    </asp:ImageMap>
```

提示：

HotSpots(作用点)的坐标以像素为单位，可以在画图或浏览器程序中打开，移动鼠标，在画图下部状态栏中可看到当前鼠标所在处的像素点的坐标，或者使用截图快捷键测量，如图 5-11 所示。

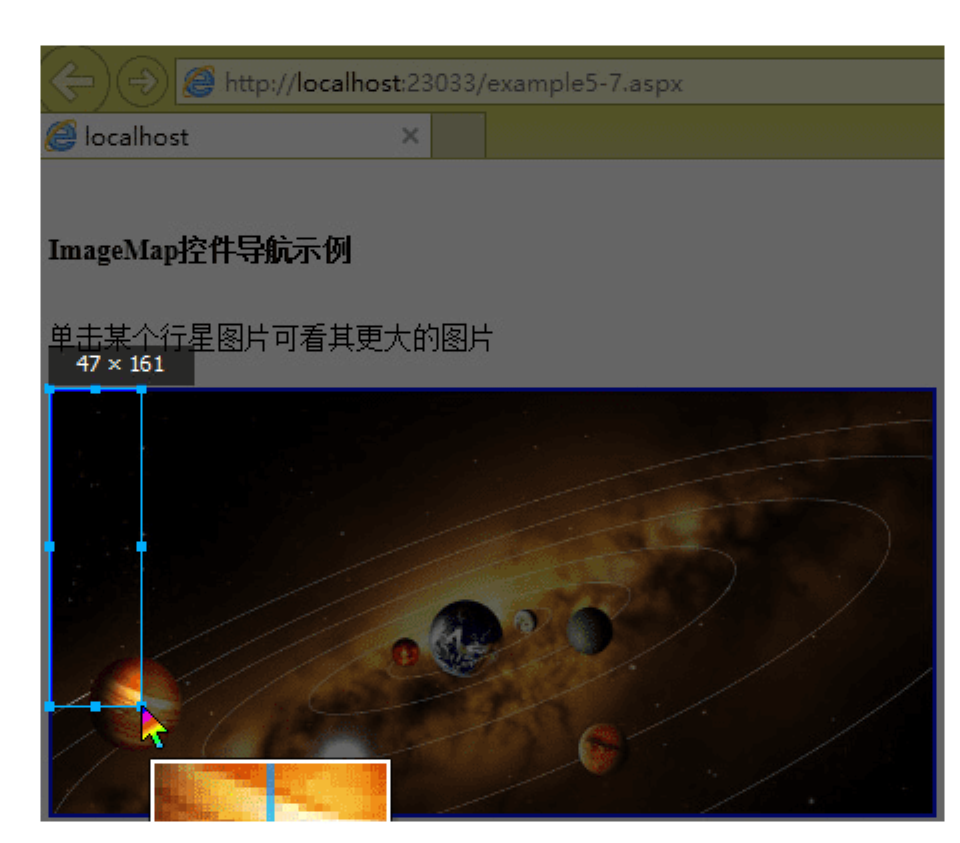


图 5-11　利用截图快捷键寻找图像当前光标处的像素点坐标

⑥ Image1 控件的属性设置为：ImageUrl="~/Images/xing1.jpg"；Image2 控件的属性设置为：ImageUrl="~/Images/xing2.jpg"。

⑦ 按 Ctrl+F5 组合键，观看显示效果。

提示：

其他行星的处理不再详细介绍，感兴趣的读者可以练习使用同样的方法做出来。

5.3.6　RadioButton 和 RadioButtonList 服务器控件

在向 ASP.NET 网页添加单选按钮时，可以使用两种服务器控件：单个 RadioButton 控件或 RadioButtonList 控件。这两种控件都允许用户从一小组互相排斥的预定义选项中进行选择。这些控件允许定义任意数目带标签的单选按钮，并将它们水平或垂直排列。

每类控件都有各自的优点。单个 RadioButton 控件使用户可以更好地控制单选按钮组的布局。例如，可以在各单选按钮之间加入文本(即非单选按钮文本)。

RadioButtonList 控件不允许在按钮之间插入文本，但如果想将按钮绑定到数据源，使用这类控件要方便得多，在编写代码以检查所选定的按钮方面，也简单一些。

1. 对单选按钮分组

单选按钮很少单独使用，而是进行分组以提供一组互斥的选项。在一个组内，每次只能选择一个单选按钮。可以用下列方法创建分组的单选按钮。

(1) 先向页面中添加单个的 RadioButton 控件，然后将所有这些控件手动分配到一个组中。具有相同组名的所有单选按钮视为单个组的组成部分。

(2) 向页面中添加一个 RadioButtonList 控件。该控件中的列表项将自动进行分组。

2. RadioButton 事件

在单个 RadioButton 控件和 RadioButtonList 控件之间，事件的工作方式略有不同。

单个 RadioButton 控件在用户单击该控件时引发 CheckedChanged 事件。默认情况下，这一事件并不导致向服务器发送页面，但通过将 AutoPostBack 属性设置为 true，可以使该控件强制立即发送。

与单个的 RadioButton 控件相反，RadioButtonList 控件在用户更改列表中选定的单选按钮时会引发 SelectedIndexChanged 事件。默认情况下，此事件并不导致向服务器发送窗体，但可以通过将 AutoPostBack 属性设置为 true 来指定此选项。

【例 5-8】演示单选按钮控件 RadioButton 和单选按钮列表控件 RadioButtonList 使用。

① 启动 VS，打开 WebSite5，新建名为 example5-8.aspx 的 ASP.NET 页面。将准备好的图片复制到 Images 文件夹中。

② 切换到 example5-8.aspx 页的【设计】视图，如图 5-12 所示。

③ 输入静态文本“请在下方输入个人信息：”，在【格式】工具栏中使用【块格式】下拉列表将该文本的格式设置为“标题 1”。

④ 输入静态文本“姓名”、“性别”、“请选择要看的行星”和 RadioButtonList Example。

⑤ 从工具栏中依次将控件拖动到设计视图，添加 TextBox1、RadioButton1 和 RadioButton2 这 3 个控件。

⑥ 在 VS 中的【表】菜单上，单击【插入表】，出现【插入表格】对话框。

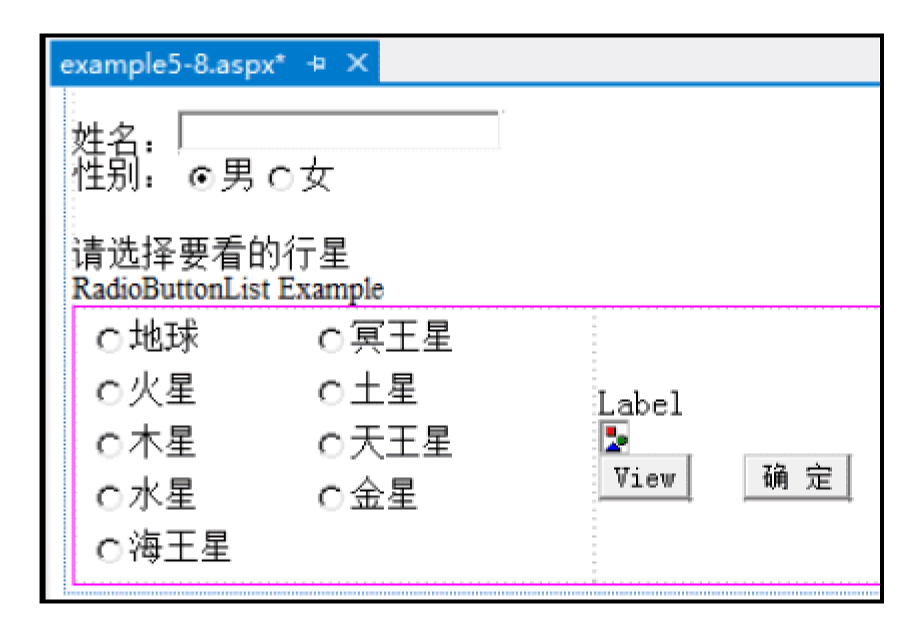


图 5-12　按钮控件

⑦ 设置行数值为 1，列数值为 2，单元格衬距值为 2，单元格间距值为 0，边框粗细值为 1，边框颜色值为#FF00FF，单击【确定】按钮，【设计】视图上出现一个 1 行 2 列的表格。

⑧ 从工具栏中依次将控件拖动到表格的各单元格中，也可先在表格的各单元格中单击，再在工具栏中相应的控件上双击添加控件。在第 1 列中添加 RadioButtonList1 控件，在第 2 列中添加 Label1、Image1、Button1 和 Button2 控件。

⑨ 属性设置如下。

RadioButton1 控件的属性设置为：ID="RadioButtonMale"、GroupName="sex"、Text="男"、Checked="True"；

RadioButton2 控件的属性设置为：ID="RadioButtonFemale"、GroupName="sex"、Text="女"；

RadioButtonList1 控件的属性设置为：id="myList"、RepeatColumns="2"；

Label1 控件的属性设置为：id="msg"；

Image1 控件的属性设置为：id="Image"；

Button1 控件的属性设置为：ID="Button1"、Text="View"；

Button2 控件的属性设置为：ID="ButtonOK"、onclick="ButtonOK_Click"、Text="确定"。

⑩ 在【设计】视图中，单击 RadioButtonList1 控件右上角的【智能标记】，切换到【RadioButtonList 任务】对话框，不选择【启用 AutoPostBack】，单击【编辑项】，出现【ListItem 集合编辑器】，如图 5-13 所示，单击【添加】，添加成员 ListItem0~ListItem8，分别将其属性 Text 值设置为“地球”、“火星”、“木星”、“水星”、“海王星”、“冥王星”、“土星”、“天王星”、“金星”，分别将其属性 Value 值设置为 dq.jpeg、mx.jpeg、hx.jpeg、sx.jpeg、hwx.jpeg、mwx.jpeg、tx.jpeg、twx.jpeg、jx.jpeg。其主要的 HTML 代码如下所示。

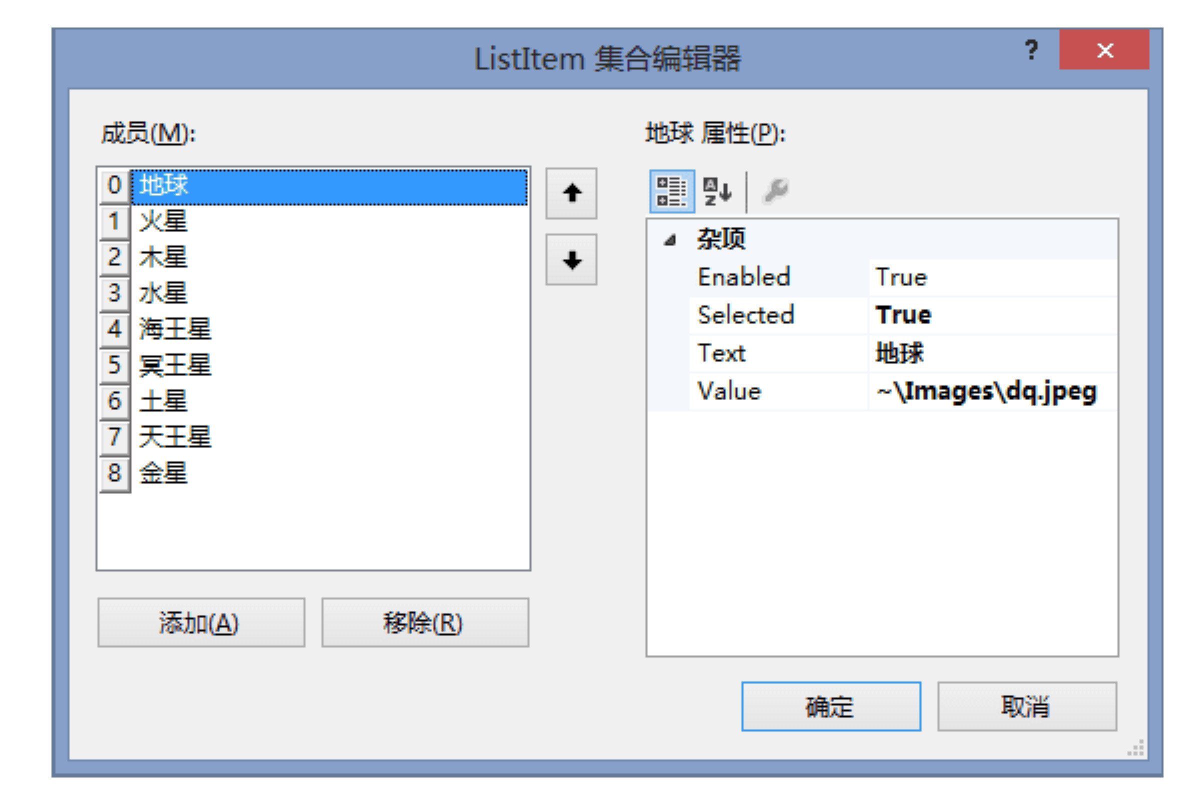


图 5-13　为 RadioButtonList 控件添加成员

```
    <asp:RadioButtonList id="myList" RepeatColumns="2" runat="server" Height="16px"
Width="204px">
        <asp:ListItem Selected="True" Value="~\Images\dq.jpeg">地球</asp:ListItem>
        <asp:ListItem Value="~\Images\hx.jpeg">火星</asp:ListItem>
        <asp:ListItem Value="~\Images\mx.jpeg">木星</asp:ListItem>
        <asp:ListItem Value="~\Images\sx.jpeg">水星</asp:ListItem>
        <asp:ListItem Value="~\Images\hwx.jpeg">海王星</asp:ListItem>
        <asp:ListItem Value="~\Images\mw.jpeg">冥王星</asp:ListItem>
        <asp:ListItem Value="~\Images\tx.jpeg">土星</asp:ListItem>
        <asp:ListItem Value="~\Images\twx.jpeg">天王星</asp:ListItem>
        <asp:ListItem Value="~\Images\jx.jpeg">金星</asp:ListItem>
     </asp:RadioButtonList>
```

⑪添加事件的处理程序代码。

```
protected void ButtonOK_Click(object sender, EventArgs e)
{
    string str1, str2, str3= "";
    if (TextBox1.Text == "")
    {
       string   scriptString = "alert('用户名不能为空！'); ";
       Page.ClientScript.RegisterClientScriptBlock(this.GetType(),"warning", scriptString, true);
    }
    else
    {
        str1 = TextBox1.Text;
        if (RadioButtonMale.Checked)
           {   str2 = "男"; }
        else
           { str2 = "女"; }
        str3 = this .myList.SelectedItem.Text;
        string scriptContent = "alert('您提供的信息是：姓   名："+str1+
             "   性别："+str2+"  选择的行星图片是："+str3+" '); ";
        Page.ClientScript.RegisterClientScriptBlock(this.GetType(),"success", scriptContent, true);
    }
}
void Page_Load ( object src, EventArgs e )
{
if (!Page.IsCallback)
{
    if (myList.SelectedIndex > -1)
    {
        msg.Text = myList.SelectedItem.Text;
        Image.ImageUrl = "~/Images/" + myList.SelectedItem.Value;
        Image.AlternateText = myList.SelectedItem.Text;
    }
  }
}
```

⑫ 按 Ctrl+F5 组合键，观看显示效果。

5.3.7　MultiView 和 Wizard 控件

1. MultiView 控件

MultiView 和 View 控件可以制作出选项卡的效果，MultiView 控件用作一个或多个 View 控件的外部容器。View 控件又可包含标记和控件的任何组合。如果要切换视图，可以使用控件的 ID 或者 View 控件的索引值。在 MultiView 控件中，一次只能将一个 View 控件定义为活动视图。如果某个 View 控件定义为活动视图，它所包含的子控件则会呈现到客户端。可以使用 ActiveViewIndex 属性或 SetActiveView 方法定义活动视图。如果 ActiveViewIndex 属性为空，则 MultiView 控件不向客户端呈现任何内容。如果活动视图设置为 MultiView 控件中不存在的 View，则会在运行时引发 ArgumentOutOfRangeException。下面介绍一些 MultiView 控件常用的属性、方法。

- ActiveViewIndex 属性：用于获取或设置当前被激活显示的 View 控件的索引值。默认值为 -1，表示没有 View 控件被激活。
- SetActiveView 方法：用于激活显示特定的 View 控件。
- ActiveViewChanged 事件：当视图切换时被激发。
- MultiView 控件一次显示一个 View 控件，并公开该 View 控件内的标记和控件。通过设置 MultiView 控件的 ActiveViewIndex 属性，可以指定当前可见的 View 控件。

(1) 呈现 View 控件内容

未选择某个 View 控件时，该控件不会呈现到页面中。但是，每次呈现页面时都会创建所有 View 控件中的所有服务器控件的实例，并且将这些实例的值存储为页面的视图状态的一部分。

无论是 MultiView 控件还是各个 View 控件，除当前 View 控件的内容外，都不会在页面中显示任何标记。例如，这些控件不会以与 Panel 控件相同的方式来呈现 div 元素。这些控件也不支持可以作为一个整体应用于当前 View 控件的外观属性。但是可以将一个主题分配给 MultiView 或 View 控件，控件将该主题应用于当前 View 控件的所有子控件。

(2) 引用控件

每个 View 控件都支持 Controls 属性，该属性包含该 View 控件中的控件集合。也可以在代码中单独引用 View 控件中的控件。

(3) 在视图间导航

除了通过将 MultiView 控件的 ActiveViewIndex 属性设置为要显示的 View 控件的索引值可以在视图间导航外，MultiView 控件还支持可以添加到每个 View 控件的导航按钮。

若要创建导航按钮，可以向每个 View 控件添加一个按钮控件(Button、LinkButton 或 ImageButton)，然后可以将每个按钮的 CommandName 和 CommandArgument 属性设置为保留值以使 MultiView 控件移动到另一个视图。

【例 5-9】View 和 MultiView 控件示例。

① 启动 VS，打开 WebSite5。

② 新建名为 example5-9.aspx 的 ASP.NET 页面。

③ 切换到 example5-9.aspx 页的【设计】视图。

④ 输入静态文本“按书名、类别名还是出版社名称进行搜索？”，如图 5-14 所示，添加 3 个 RadioButton 控件到页面上。

图 5-14　添加控件

切换到【源】视图，修改其 HTML 代码为：

```
<asp:RadioButton ID="radioProduct"    runat="server"    autopostback="true"
    GroupName="SearchType"    Text="书名"
    OnCheckedChanged="radioButton_CheckedChanged" />  
<asp:RadioButton ID="radioCategory"    runat="server"    autopostback="true"
    GroupName="SearchType"    Text="类别"
    OnCheckedChanged="radioButton_CheckedChanged" />
<asp:RadioButton ID="radioPublisher"    runat="server"    AutoPostBack="True"
    GroupName="SearchType"    Text="出版社"
    oncheckedchanged="radioButton_CheckedChanged" />
```

请注意将 3 个 RadioButton 的 CheckChanged 事件的处理程序设置为 oncheckedchanged= "radioButton_CheckedChanged"，这样单击任一 RadioButton，响应它们的处理程序是相同的。

⑤ 如图 5-14 所示，从工具箱的【标准】选项卡中，拖动 MultiView 控件到页面上，再拖动 3 个 View 控件到 MultiView 上，拖动一个 Button 控件到页面上。

分别单击3个View控件，将其ID属性分别改为viewProductSearch、ViewCategorySearch、ViewPublisher；分别直接输入静态文本“输入书名”、“输入类别”、“输入出版社名”；从工具箱的“标准”选项卡中，拖动3个Textbox控件分布到3个View控件上，将其ID属性分别改为textProductName、textCategory、textPublisher。

⑥ 切换到【源】视图中，可看到如下所示的代码。

```
<asp:MultiView ID="MultiView1" runat="server"> <br />
    <asp:View ID="viewProductSearch" runat="server">
        输入书名：  <asp:TextBox ID="textProductName" runat="server"> </asp:TextBox>
    </asp:View> <br />
    <asp:View ID="viewCategorySearch" runat="server">
        输入类别：<asp:TextBox ID="textCategory" runat="server"> </asp:TextBox>
    </asp:View> <br />
    <asp:View ID="ViewPublisher" runat="server">
        输入出版社名：<asp:TextBox ID="textPublisher" runat="server"></asp:TextBox>
    </asp:View>
</asp:MultiView>
```

⑦ 设置 Button1 控件的标记如下所示。

```
<asp:Button ID="btnSearch" OnClick="Button1_Click" runat="server" Text="Search" />
```

⑧ 切换到 MultiViewControl.aspx.cs，在【类】体内添加如下所示的代码。

```
public enum SearchType
{
    NotSet = -1,
    Products = 0,
    Category = 1,
    Publisher = 2
}
protected void Page_Load(object sender, EventArgs e)
{
    radioProduct.Checked = true;
    MultiView1.ActiveViewIndex = 0;
}
protected void Button1_Click(Object sender, System.EventArgs e)
{
    if (MultiView1.ActiveViewIndex > -1)
    {
        SearchType mSearchType = (SearchType)MultiView1.ActiveViewIndex;
        switch (mSearchType)
        {
            case SearchType.Products:
                DoSearch(textProductName.Text, mSearchType);
                break;
            case SearchType.Category:
                DoSearch(textCategory.Text, mSearchType);
                break;
            case SearchType.Publisher:
                DoSearch(textPublisher.Text, mSearchType);
                break;
            case SearchType.NotSet:
                break;
        }
    }
}

protected void DoSearch(String searchTerm, SearchType type)
{
    // Code here to perform a search.
    string scriptString = "alert('"+"您输入的"+searchTerm+"');";
    Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "success", scriptString, true);
    // Response.Write("您输入的"+ searchTerm );
}

protected void radioButton_CheckedChanged(Object sender, System.EventArgs e)
{
    if (radioProduct.Checked)
    {
        MultiView1.ActiveViewIndex = (int)SearchType.Products;
    }
```

```
        else if (radioCategory.Checked)
        {
            MultiView1.ActiveViewIndex = (int)SearchType.Category;
        }
        else if (radioPublisher.Checked)
        {
            MultiView1.ActiveViewIndex = (int)SearchType.Publisher;
        }
    }
```

⑨ 按 Ctrl+F5 组合键，观看程序运行效果。

2. Wizard 控件

Wizard 控件如其名所示，是一个用于创建向导的复杂用户控件。与 MultiView 控件类似，使用该控件也可以用于将一个大的表单分割为多个子表单。但是 Wizard 控件包含更多的功能和定义行为，如包含导航按钮、导航工具条、步骤链接，以及样式和模板等。

(1) 创建一个名为 WizardControlDemo 的网站，在 Default.aspx 页面中添加一个 Wizard 控件。默认情况下，该控件提供一个导航按钮和一个在左侧具有导向链接的侧边栏，VS 生成的代码如下。

```
<asp:Wizard ID="Wizard1" runat="server">
<WizardSteps>
    <asp:WizardStep runat="server" title="Step 1">
    </asp:WizardStep>
    <asp:WizardStep runat="server" title="Step 2">
    </asp:WizardStep>
</WizardSteps>
</asp:Wizard>
```

(2) 在 Wizard 控件中，每个向导页由一个 WizardStep 组成。Wizard 控件包含一个 WizardStepCollection 控件的集合，其中包含由开发人员定义的每个步骤的用户界面。开发人员可以使用该控件的智能任务面板添加 WizardStep，或在属性窗格中选中 WizardSteps，单击右侧的按钮，打开【WizardStep 集合编辑器】对话框，如图 5-15 所示。

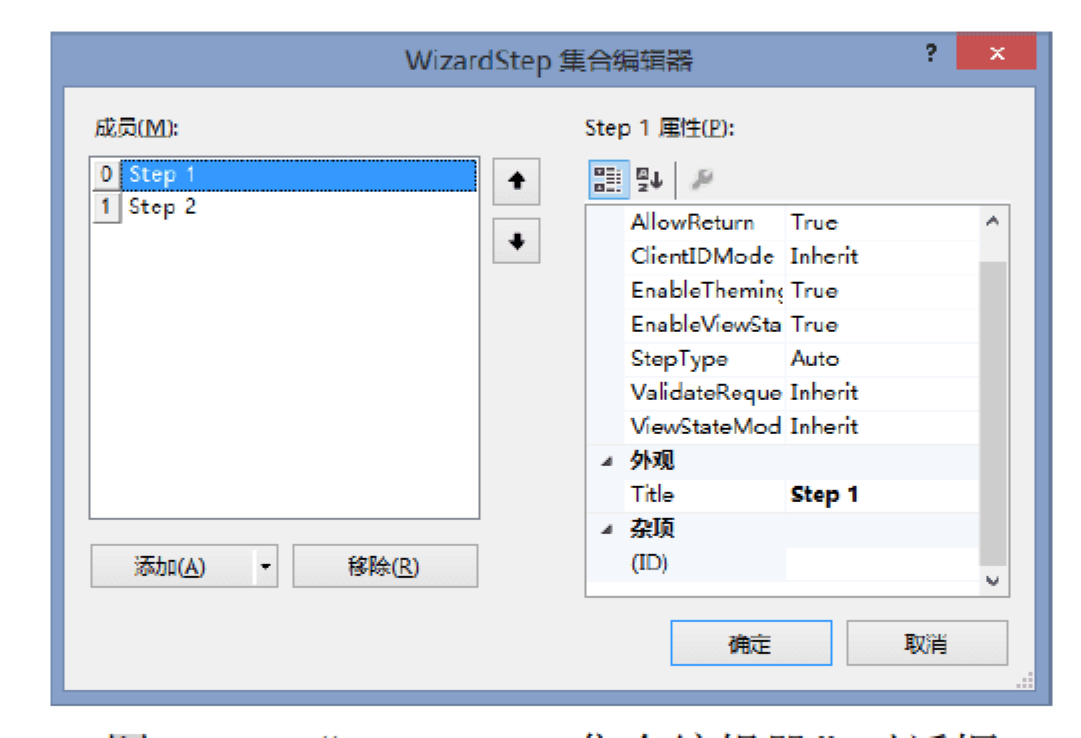


图 5-15　“WizardStep 集合编辑器”对话框

在【WizardStep 集合编辑器】对话框中，可以添加或移除指定类型的 WizardStep 对象，设置步骤的属性。可以看到 WizardStep 提供了如下几个属性以供设置。

- Title：提供步骤的描述性名称，它将作为一个链接显示在向导控件左边的 SlideBar 栏上。

- StepType：这是一个 WizardStepType 枚举类型的属性值，用于指定步骤的类型。该属性值将决定导航按钮的类型。可选值有 Start，显示一个【下一步】按钮；Step，显示【上一步】和【下一步】按钮；Finish，显示【完成】和【上一步】按钮；Complete，不显示按钮，并且隐藏 SideBar；Auto，根据在集合中的上下位置进行推断类型，默认值为 Auto。
- AllowReturn：指定用户是否能返回到这一步骤。一旦用户经过了这一步，将不能再跳回到这步，SideBar 的链接也不受影响。

Wizard 控件可以通过线性导航或非线性导航收集数据。跳过不需要的步骤或返回前面已完成的步骤以更改某个值，这种行为就属于非线性导航。Wizard 控件维持它在各步骤之间的状态，以便在所有步骤完成之前，无须将某个步骤中输入的数据保存到数据存储区中。

【例 5-10】WizardStep 中添加 Web 服务器控件，本节将创建一个简单的用户注册向导。

① 启动 VS，打开 WebSite5。

② 在页面中添加一个 Wizard 控件，并设置其属性，显示效果如图 5-16 所示。示例代码如下：

```
<form id="form1" runat="server">
<div>
<asp:Wizard id="Wizard1" runat="server" ActiveStepIndex="0" DisplayCancelButton="true">
 <WizardSteps>
 <asp:WizardStep runat="server" id="Step1" title="第一步，输入基本信息" StepType="Start">
 <b>请输入您的个人基本信息：</b>
 <br />
 请输入您的姓名：<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
 <br/>
 请输入您的 E-mail：<asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
 </asp:WizardStep>
<asp:WizardStep runat="server" id="Step2" title="第二步，输入用户信息" StepType="Step">
 <b>请输入您的个人用户信息：</b>
 <br />
 用户名：<asp:TextBox ID="TextBox3" runat="server"></asp:TextBox>
 <br/>
  密码：<asp:TextBox ID="TextBox4" runat="server"></asp:TextBox>
</asp:WizardStep>
 <asp:WizardStep runat="server" StepType="Finish" ID="Finish" Title="第三步，信息汇总">
 <b>登录信息汇总：</b><br/>
 你的姓名是：<asp:Label ID="Label1" runat="server" Text="Label" ></asp:Label><br/>
 你的 E-mail 是：<asp:Label ID="Label2" runat="server" Text="Label" ></asp:Label><br/>
 你的用户名是：<asp:Label ID="Label3" runat="server" Text="Label" ></asp:Label><br/>
 你的密码是：<asp:Label ID="Label4" runat="server" Text="Label" ></asp:Label><br/>
      </asp:WizardStep>
    </WizardSteps>
  </asp:Wizard>
 </div>
</form>
```

Wizard 控件的 ActiveStepIndex 属性设置为 0，表示将显示第一个 WizardStep。在上述

的 3 个 WizardStep 中，每个都具有不同的 StepType。第一个设置为 Start，将只显示【下一步】按钮，而不显示【上一步】按钮；第二个设置为 Step，将显示用于导航的【上一步】和【下一步】按钮，该步骤是介于第一个步骤和最后一个步骤之间的任意步骤；第三个设置为 Finish，将显示用于导航的【完成】按钮，该步骤是收集用户数据的最后一个步骤。用户可以单击【上一步】和【下一步】按钮在多个向导页之间进行导航，也可以单击侧边栏的链接来进行导航。

本示例的设计效果如图 5-16 所示。

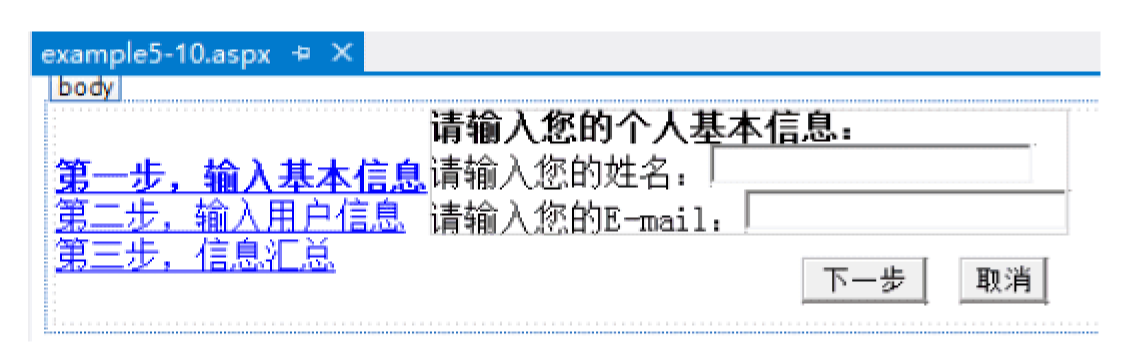


图 5-16　Wizard 控件使用演示

5.3.8　AdRotator(广告控件)服务器控件

AdRotator服务器控件提供一种在ASP.NET网页上显示广告的方法。该控件可显示.gif文件或其他图形图像。当用户单击广告时，系统会重定向到指定的目标URL。

AdRotator 服务器控件可从数据源(通常是XML 文件或数据库表)提供的广告列表中自动读取广告信息，如图形文件名和目标 URL。用户可以将信息存储在一个 XML 文件或数据库表中，然后将 AdRotator 控件绑定到该文件。

AdRotator 控件会随机选择广告，每次刷新页面时都将更改显示的广告。广告可以加权以控制广告条的优先级别，这可以使某些广告的显示频率比其他广告高。也能编写在广告间循环的自定义逻辑。

AdRotator 控件的所有属性都是可选的。XML 文件中可以包括下列属性。

- ImageUrl：要显示图像的 URL。
- NavigateUrl：单击 AdRotator 控件时要转到网页的 URL。
- AlternateText：图像不可用时显示的文本。
- Keyword：可用于筛选特定广告的广告类别。
- Impressions：一个指示广告的可能显示频率的数值(加权数值)。在 XML 文件中，所有 Impressions 值的总和不能超过 2，048，000，000 - 1。
- Height：广告的高度(以像素为单位)。此值会重写 AdRotator 控件的默认高度设置。
- Width：广告的宽度(以像素为单位)。此值会重写 AdRotator 控件的默认宽度设置。

【例 5-11】使用 AdRotator 服务器控件显示数据库中的广告。

(1) 启动 VS，打开 WebSite5，新建名为 example5-11.aspx 的网页。

(2) 将准备好的广告图片复制到 Images 文件夹。

(3) 在网站的 App_Data 文件夹中新建名为 AdDataSource.ads 的 XML 文件。

为了更好地保证安全，请将该文件的文件扩展名命名为除.xml 之外的名称，如.ads。建议将广告文件放置在 App_Data 文件夹中，因为 ASP.NET 可防止浏览器利用该文件夹中

的文件。

(4) 向广告信息文件中添加下列 XML 元素。

```
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements xmlns="http://schemas.microsoft.com/AspNet/AdRotator-Schedule-File">
</Advertisements>
```

(5) 在 Advertisements 元素中，为每个要包括在广告列表中的广告创建一个 Ad 元素，然后保存文件。

```
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements xmlns="http://schemas.microsoft.com/AspNet/AdRotator-Schedule-File">
  <Ad>
    <ImageUrl>~/Images/baidu.jpg</ImageUrl>
    <NavigateUrl>http://www.baidu.com</NavigateUrl>
    <AlternateText>Ad for baidu, Ltd. Web site</AlternateText>
    <Impressions>100</Impressions>
  </Ad>
  <Ad>
    <ImageUrl>~/Images/google.jpg</ImageUrl>
    <NavigateUrl>http://www.google.com</NavigateUrl>
    <AlternateText>Ad for Google Web site</AlternateText>
    <Impressions>50</Impressions>
  </Ad>
</Advertisements>
```

(6) 切换到 example5-11.aspx，在网页上要显示广告的位置放置一个 AdRotator，如图 5-17 所示。

(7) 单击 AdRotator 控件的【智能标记】，选择【新建数据源】，选择【xml 文件】，单击【确定】，在【配置数据源】对话框中，将【数据文件】输入框设置为 ~/App_Data/AdDataSource.ads，最后单击【确定】。

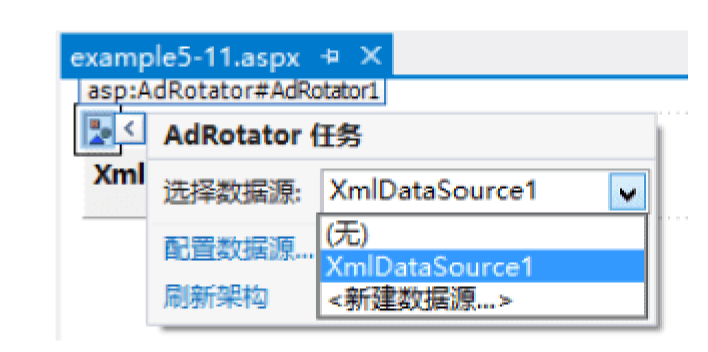


图 5-17　AdRotator 控件

(8) 测试广告。按 Ctrl+F5 运行该页。单击几次浏览器的【刷新】按钮可显示不同的广告信息。出现的广告是随机变化的。

5.3.9　Literal 控件和 Panel 控件

Literal 控件和 Panel 控件均可作为容器控件，但二者的适用场合不同，下面分别介绍。

1. Literal 控件

Literal 控件可以作为页面上其他内容的容器，最常用于向页面中动态添加内容。

对于静态内容，无须使用容器，可以将标记作为 HTML 直接添加到页面中。但是，如果要动态添加内容，则必须将内容添加到容器中。典型的容器有 Label 控件、Literal 控件、Panel 控件和 PlaceHolder 控件。

Literal 控件与 Label 控件的区别在于 Literal 控件不向文本中添加任何 HTML 元素。

(Label 控件呈现一个 span 元素。)因此，Literal 控件不支持包括位置属性在内的任何样式属性。但是，Literal 控件允许指定是否对内容进行编码。

Panel 和 PlaceHolder 控件呈现为 div 元素，这将在页面中创建离散块，与 Label 和 Literal 控件进行内嵌呈现的方式不同。

通常情况下，当希望文本和控件直接呈现在页面中而不使用任何附加标记时，可使用 Literal 控件。

Literal 控件常用的属性是 Mode 属性，该属性用于指定控件对用户所添加的标记的处理方式。可以将 Mode 属性设置为以下值。

- Transform.：将对添加到控件中的任何标记进行转换，以适应请求浏览器的协议。如果向使用 HTML 外的其他协议的移动设备呈现内容，此设置非常有用。
- PassThrough.：添加到控件中的任何标记都将按原样呈现在浏览器中。
- Encode：将使用 HtmlEncode 方法对添加到控件中的任何标记进行编码，这会将 HTML 编码转换为其文本表示形式。例如，<b>标记将呈现为<b>。当希望浏览器显示而不解释标记时，编码将很有用。编码对于安全也很有用，有助于防止在浏览器中执行恶意标记。显示来自不受信任的源的字符串时推荐使用此设置。

【例 5-12】 演示 Literal 控件的使用。

(1) 启动 VS，打开 WebSite5，新建一个名为 example5-12.aspx 的网页。

(2) 向 example5-12.aspx 页面添加 2 个 RadioButton 控件，1 个 Literal 控件，1 个 Label 控件。按如下所示的标记代码设置 2 个 RadioButton 控件的相关属性。

```
<asp:RadioButton ID="radioEncode" runat="server" GroupName="LiteralMode" Checked="True"
    Text="Encode" AutoPostBack="True" />
<asp:RadioButton ID="radioPassthrough" runat="server" GroupName="LiteralMode"
    Text="PassThrough" AutoPostBack="True" />
```

(3) 添加 Page_Load 事件代码。

```
protected void Page_Load(object sender, EventArgs e)
{
    Literal1.Text = "This <b>text</b> is inserted dynamically.";
    Label1 .Text ="This <b>text</b> is inserted dynamically.";
    if (radioEncode.Checked == true)
    {
        Literal1.Mode = LiteralMode.Encode;
    }
    if (radioPassthrough.Checked == true)
    {
        Literal1.Mode = LiteralMode.PassThrough;
    }
}
```

(4) 按 Ctrl+F5 组合键运行该页，效果如图 5-18 所示。选择不同的单选按钮，观察文本的变化。

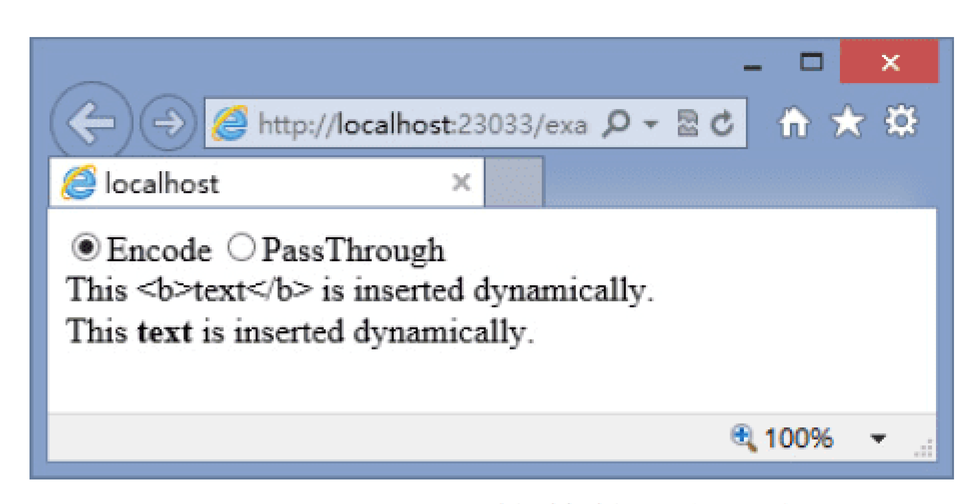


图 5-18　Literal 控件的运行示例

2. Panel 控件

Panel 控件在 ASP.NET 网页内提供了一种容器控件，可以将它用作静态文本和其他控件的父控件，向该控件添加其他控件和静态文本。

可以将 Panel 控件用作其他控件的容器。当以编程方法创建内容并需要一种将内容插入到页面中的方法时，此方法尤为适用。以下部分描述了可以使用 Panel 控件的其他方法。

(1) 动态生成的控件的容器

Panel 控件为在运行时创建的控件提供了一个方便的容器。

(2) 对控件和标记进行分组

对于一组控件和相关的标记，可以通过把其放置在 Panel 控件中，然后操作此 Panel 控件的方式将它们作为一个单元进行管理。例如，可以通过设置面板的 Visible 属性来隐藏或显示该面板中的一组控件。

(3) 具有默认按钮的窗体

可将 TextBox 控件和 Button 控件放置在 Panel 控件中，然后通过将 Panel 控件的 DefaultButton 属性设置为面板中某个按钮的 ID 来定义一个默认的按钮。如果用户在面板内的文本框中进行输入时按 Enter，这与用户单击特定的默认按钮具有相同的效果。这有助于用户更有效地使用项目窗体。

(4) 向其他控件添加滚动条

有些控件(如 TreeView 控件)没有内置的滚动条。通过在 Panel 控件中放置滚动条控件，可以添加滚动行为。若要向 Panel 控件添加滚动条，请设置 Height 和 Width 属性，将 Panel 控件限制为特定的大小，然后再设置 ScrollBars 属性。

(5) 页上的自定义区域

可使用 Panel 控件在页面上创建具有自定义外观和行为的区域，如下所示。

- 创建一个带标题的分组框：可设置 GroupingText 属性来显示标题。呈现页时，Panel 控件的周围将显示一个包含标题的框，其标题是用户指定的文本。

说明：

不能在 Panel 控件中同时指定滚动条和分组文本。如果设置了分组文本，其优先级高于滚动条。

- 在页面上创建具有自定义颜色或其他外观的区域：Panel 控件支持外观属性(如 BackColor 和 BorderWidth)，可以设置外观属性为页面上的某个区域创建独特的外观。

说明：

设置 GroupingText 属性将自动在 Panel 控件周围呈现一个边框。

【例 5-13】演示文件 Panel 控件的使用。

① 启动 VS，打开 WebSite5，新建名为 example5-13.aspx 的网页。

② 新建页面 example5-13.aspx，切换到其【设计】视图。

③ 从工具箱的【标准】选项卡中，将 Panel 控件拖动到页面上，如图 5-19 所示。

④ 添加 DropDownList1、CheckBox1、CheckBox2、Button1、Literal1、Literal2 控件，如图 5-19 所示。

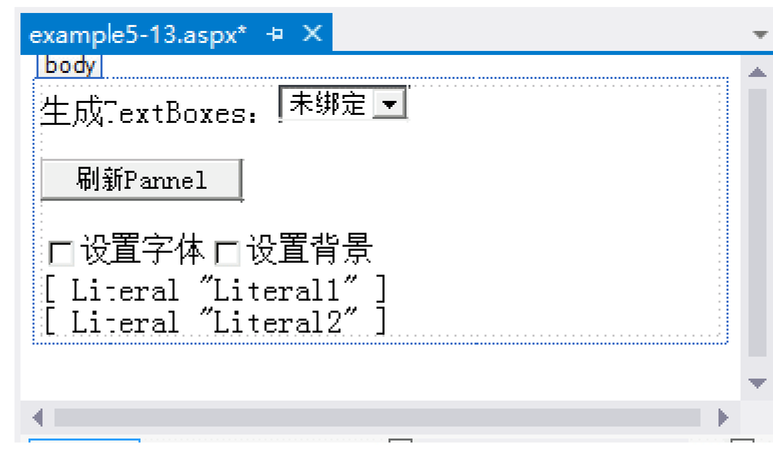


图 5-19　添加 Panel 等控件

⑤ 切换到 example5-13.aspx 页的【源】视图，根据下面的代码设置各个控件的属性。

```
<asp:panel id="myPanel" runat="server" backcolor="#eeeeee" width="300px"
        GroupingText="Panel 控件">
        <p>作为动态生成的文本框的容器…… </p>
</asp:panel>
生成 TextBoxes:
<asp:dropdownlist id="DropDownList1" runat="server">
    <asp:listitem value="1">1</asp:listitem>
    <asp:listitem value="2">2</asp:listitem>
    <asp:listitem value="3">3</asp:listitem>
</asp:dropdownlist>
<asp:button ID="Button1" text="刷新 Panel" runat="server" onclick="Button1_Click" />
<asp:CheckBox ID="CheckBoxChangeFont" runat="server" AutoPostBack="True"
        oncheckedchanged="CheckBoxChangeFont_CheckedChanged" Text="设置字体"/>
<asp:CheckBox ID="CheckBoxChangeBkGround" runat="server" AutoPostBack="True"
    oncheckedchanged="CheckBoxChangeBkGround_CheckedChanged" Text="设置背景"/>
```

⑥ 切换到 example5-13.aspx.cs 页的【代码】视图，根据下面的代码设置各控件的 Click 事件的处理程序。

```
void Page_Load(object src, EventArgs e)
{
        // generate textbox controls
        int numtexts = int.Parse(DropDownList1.SelectedItem.Value);
        for (int i = 1; i <= numtexts; i++)
        {
            myPanel.Controls.Add(new LiteralControl("<p>"));
            TextBox t = new TextBox();
            t.Text = "TextBox" + i.ToString();
            t.ID = "TextBox" + i.ToString();
            myPanel.Controls.Add(t);
        }
}
protected void CheckBoxChangeFont_CheckedChanged(object sender, EventArgs e)
{
```

```
        if (CheckBoxChangeFont.Checked)
        {
            this.myPanel.Font.Italic = true;
            this.myPanel.ForeColor = System.Drawing.Color.Red ;
            Literal1 .Text ="当前所显示字型是“斜体”，颜色是“红色”";
        }
        else
        {
            this.myPanel.Font.Italic = false ;
            this.myPanel.ForeColor = System.Drawing.Color.Blue;
            Literal1.Text = "当前所显示字型是“默认字体”，颜色是“蓝色”";
        }
    }
    protected void CheckBoxChangeBkGround_CheckedChanged(object sender, EventArgs e)
    {
        if (CheckBoxChangeBkGround.Checked)
        {
            this.myPanel.BackColor    = System.Drawing.Color.Bisque;//Bisque 橘黄色
            Literal2.Text = "当前所显示背景颜色是“Bisque 橘黄色”。";
        }
        else
        {
            this.myPanel.BackColor = System.Drawing.Color.Beige;//Beige  米黄色
            Literal2.Text = "当前所显示背景颜色是“Beige  米黄色”。";
        }
    }
```

⑦ 按 Ctrl+F5 组合键，观看显示效果。

5.4　验证控件

ASP.NET 为开发人员提供了一套完整的服务器控件来验证用户输入的信息是否有效，这些控件可与 ASP.NET 网页上的任何控件(包括 HTML 和服务器控件)一起使用。

5.4.1　ASP.NET 验证控件

表 5-2 列出了 ASP.NET 验证控件及其功能说明。

表 5-2　ASP.NET 验证控件

验证类型	使用的控件	说　　明
必选项	必选项验证控件 RequiredFieldValidator	验证一个必填字段，如果这个字段没填，那么，将不能提交信息
与某值的比较	比较验证控件 CompareValidator	比较验证：将用户输入与一个常数值或者另一个控件或特定数据类型的值进行比较(使用小于、等于或大于等比较运算符)，同时也可以用来校验控件中内容的数据类型：如整型、字符串型等。如密码和确认密码两个字段是否相等

(续表)

验证类型	使用的控件	说　明
范围检查	范围验证控件 RangeValidator	范围验证：RangeValidator 控件可以用来判断用户输入的值是否在某一特定范围内。可以检查数字对、字母对和日期对限定的范围。属性 MaximumValue 和 MinimumValue 用来设定范围的最大和最小值
模式匹配	正则表达式验证控件 RegularExpressionValidator	正则表达式验证：它根据正则表达式来验证用户输入字段的格式是否合法，如电子邮件、身份证、电话号码等。ControlToValidate 属性选择需要验证的控件，ValidationExpression 属性则编写需要验证的表达式的样式
用户定义	用户定义验证控件 CustomValidator	使用自己编写的验证逻辑检查用户输入。此类验证能够检查在运行时派生的值。在运行定制的客户端 JavaScript 或 VBScript 函数时，可以使用这个控件
验证汇总	验证汇总控件 ValidationSummary	验证汇总：该控件不执行验证，但该控件将本页所有验证控件的验证错误信息汇总为一个列表并集中显示，列表的显示方式由 DisplayMode 属性设置

- 验证对象模型：通过使用由各个验证控件和页面公开的对象模型，可以与验证控件进行交互。每个验证控件都会公开自己的 IsValid 属性，可以测试该属性以确定该控件是否通过验证测试。页面还公开一个 IsValid 属性，该属性总结页面上所有验证控件的 IsValid 状态。该属性允许开发者执行单个测试，以确定是否可以继续自行处理。页面还公开一个包含页面上所有验证控件的列表的 Validators 集合。可以依次通过这一集合来检查单个验证控件的状态。
- Button 控件和验证：如果某一页面包含 ASP.NET 验证控件，则在默认情况下，单击按钮控件会导致验证控件执行检查。如果为验证控件启用了客户端验证，则在验证检查失败时不会提交该页面。表 5-3 描述了按钮控件所支持的可以更精确地控制验证过程的属性。

表 5-3　Button 控件的部分属性

属　性	说　明
CausesValidation	指定单击按钮是否还执行验证检查。将此属性设置为 false 可避免验证检查
ValidationGroup	能够指定单击按钮时调用页面上的哪些验证程序。如果未建立任何验证组，则单击按钮会调用页面上的所有验证程序

5.4.2　验证 Web 窗体页中的用户输入

本示例演示如何使用 ASP.NET 验证控件检查网页中的用户输入。

【例 5-14】验证 Web 窗体页中的用户输入，【设计】视图如图 5-20 所示。其中，电子邮件地址必须有必填验证和格式验证；预定人数有必填验证和数据范围验证；预定日期有自定义的日期格式验证。

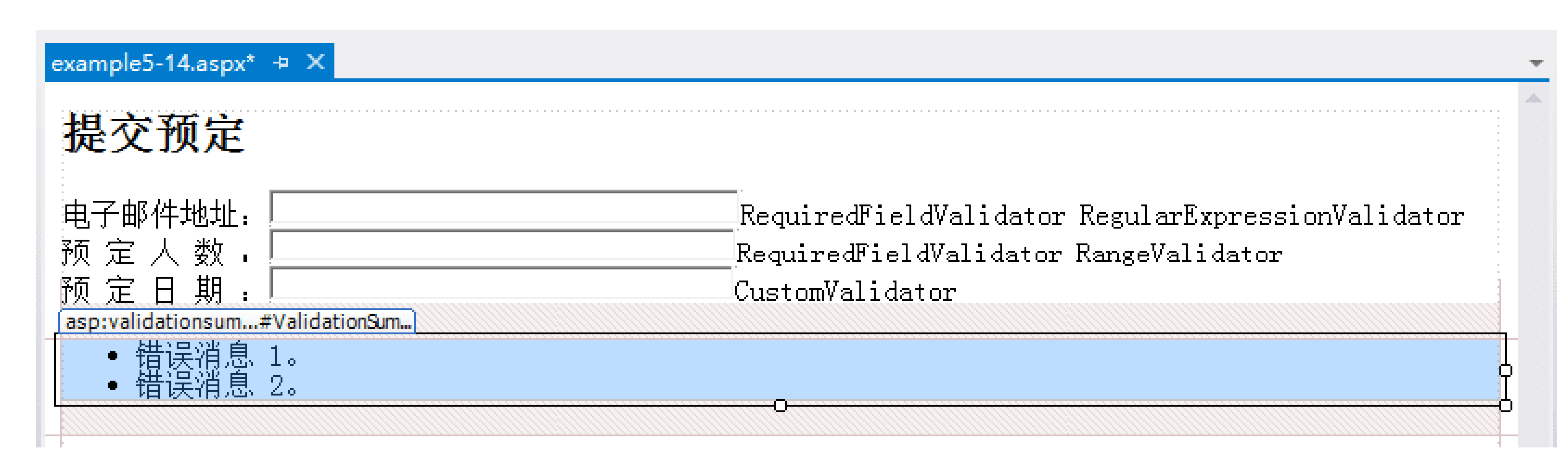


图 5-20　添加验证控件

(1) 启动 VS，打开 WebSite5，新建网页 example5-14.aspx。

(2) 切换到 example5-14.aspx 的【设计】视图，输入页标题“提交预定”。

添加3个TextBox控件，ID属性分别改为textEmail、textNumberInParty和textPreferredDate；添加Button控件，属性设置为ID="buttonSubmit"、Text="提交申请"、ValidationGroup="AllValidators"；添加Label控件，属性设置为ID="labelMessage"、Text=""。

(3) 如图 5-20 所示，在上述文本框左侧输入文本作为标题。

(4) 双击【提交申请】按钮，为其 Click 事件创建一个处理程序，添加以下代码。

```
protected void buttonSubmit_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    {
        labelMessage.Text = "您的预订已处理！";
         labelMessage.ForeColor = System.Drawing.Color.Red;
    }
}
```

该按钮处理程序只显示一条消息，它不执行实际的处理操作。但显示该消息将允许在本示例的后面部分测试验证结果。

(5) 切换到【设计】视图，双击页面空白区域以创建一个 Page_Load 处理程序，添加以下代码。

```
protected void Page_Load(object sender, EventArgs e)
{
    labelMessage.Text = "";
}
```

在通过所有验证后，该代码将清除该页上所显示的消息。该代码对本示例后面部分的测试很有用。

(6) 添加 RequiredFieldValidator 并将该控件放在 textEmail 文本框右侧，设置 RequiredFieldValidator控件的以下属性，如表 5-4 所示。

表 5-4　RequiredFieldValidator 控件的属性设置

属　性	值	说　明
ControlToValidate	textEmail	将该验证控件绑定到要验证其内容的文本框
Display	Dynamic	指定只在显示错误需要时才呈现该控件
ErrorMessage	须填电子邮件地址	显示摘要错误中的文本

(续表)

属　性	值	说　明
Text	*	星号表示某字段是必填的。只在出现错误时，才显示该文本
ValidationGroup	AllValidators	将验证控件分组

(7) 添加 RegularExpressionValidator 控件并将其放在刚添加的 RequiredFieldValidator 右侧，设置 RegularExpressionValidator 控件属性，该标签代码如下。

```
<asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat="server" ErrorMessage="邮件地址错误！" ControlToValidate="textEmail" CssClass="auto-style1" Display="Dynamic" ValidationGroup="AllValidators" ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">无效格式！</asp:RegularExpressionValidator>
```

(8) 在 RegularExpressionValidator 正则表达式验证控件仍处于选定状态时，在【属性】窗口中单击 ValidationExpression 框中的省略号按钮，在【标准表达式】列表中，单击【Internet 电子邮件地址】，单击【确定】关闭对话框。

(9) 添加 RequiredFieldValidator 控件，将该控件绑定到 textNumberInParty 文本框并将其 ErrorMessage 属性设置为【请指出团队人数】。

(10) 添加 RangeValidator 控件并将其放在刚添加的 RequiredFieldValidator 控件右侧，设置 RangeValidator 控件的属性，标签代码如下。

```
<asp:RangeValidator ID="RangeValidator1" runat="server" ControlToValidate="textNumberInParty" Display="Dynamic" MaximumValue="20" MinimumValue="1" ErrorMessage="为团队人数输入一个介于 1 和 20 之间的数字" ValidationGroup="AllValidators" Type="Integer" Text="输入一个介于 1 和 20 之间的数字"></asp:RangeValidator>
```

RangeValidator 控件执行下面两个功能：首先确保用户所输入的数据是数字，然后检查该数字是否在指定的最小值和最大值之间。

(11) 按 Ctrl+F5 组合键运行该页。分别输入正确的、不正确的电子邮件地址，在指定范围内和不在指定范围内的数字，进行验证。

(12) 从工具箱的【验证】组中，将 ValidationSummary 控件拖动到页面上，将其 ValidationGroup 属性设置为 AllValidators。运行该页，执行相同的测试。

对于每条错误，将看到在两个位置中显示了错误信息。在验证控件所在的位置上显示了一条简短的错误信息(验证程序的 Text 属性值)，在 ValidationSummary 控件中显示了一条较长的错误信息(该控件的 ErrorMessage 属性值)。

(13) 将 ValidationSummary 控件的 ShowMessageBox 属性设置为 true。运行该页。执行相同的测试。此时，每条错误将会导致在浏览器中显示一个弹出消息框。

(14) 添加 CustomValidator(自定义验证控件)放在 textPreferredDate 文本框右侧，并设置 CustomValidator 控件属性，标签代码如下。

```
<asp:CustomValidator ID="CustomValidator1" runat="server" ErrorMessage="以 yyyy-m-d 格式输入
```

```
一个日期&quot;" ControlToValidate="textPreferredDate" CssClass="auto-style1" Display="Dynamic"
ValidationGroup="AllValidators"  EnableClientScript="False" >无效日期格式(需要
yyyy-m-d)</asp:CustomValidator>
```

(15) 双击 CustomValidator 控件，为其 ServerValidate 事件创建一个处理程序，添加以下代码。

```
protected void CustomValidator1_ServerValidate(object source,ServerValidateEventArgs args)
{
   try
   {
       DateTimeFormatInfo info1 = (DateTimeFormatInfo)
            Thread.CurrentThread.CurrentCulture.GetFormat(typeof(DateTimeFormatInfo));
       DateTime time1 = DateTime.ParseExact(args.Value, "d", info1,
            DateTimeStyles.AllowWhiteSpaces);
       args.IsValid = true;
   }
    catch
    {
        args.IsValid = false;
     }
}
```

在用户提交此页面时运行该代码。该代码使用传入处理程序的 ServerValidateEventArgs (args)对象与该验证控件交互。用户在 textPreferredDate 文本框中所输入的值作为 args 对象的 Value 属性传递。在检查用户输入是否有效后，将 args 对象的 IsValid 属性相应地设置为 true 或 false。如果将该属性设置为 false，则该验证程序将显示其错误信息。

注意：

需要引入两个命名空间：System.Globalization 和 using System.Threading。

(16) 在 Button_Click 处理程序中，添加以下代码。

```
protected void buttonSubmit_Click(object sender, EventArgs e)
{
     if(Page.IsValid)
     {
          labelMessage.Text = "您的预订已受理。";
          //Your reservation has been processed.
     }
     else
          labelMessage.Text = "页面不合语法。";//Page is not valid.
     }
}
```

在使用时，必须在服务器端的任何处理期间都检查 Page.IsValid 属性以确定是否通过了所有验证检查。IsValid 属性返回该页面上所有验证控件的累积状态。此属性用于确保任何服务器端的处理逻辑都已通过所有验证检查。

(17) 按 Ctrl+F5 组合键运行该页，测试 CustomValidator 控件。

【例 5-15】演示 CompareValidator 比较验证控件的使用方法。

(1) 启动 VS，打开 WebSite5，新建名字为 example5-15.aspx 的网页。

(2) 在设计【视图】中，添加如图 5-21 所示的控件，2 个 TextBox 控件，2 个 RequiredFieldValidator控件，1个CompareValidator控件。



图 5-21　CompareValidator 控件

(3) 在 example5-15.aspx 生成如下代码。

```
*密码：
<asp:TextBox ID="TextBoxPassword"   runat="server"></asp:TextBox>
<asp:RequiredFieldValidator ID="RequiredFieldValidator3" runat="server"
     ControlToValidate="TextBoxPassword" ErrorMessage="密码不能为空！"
     Display="Dynamic">密码不能为空！
</asp:RequiredFieldValidator> <br />
*确认密码：
<asp:TextBox ID="TextBoxPasswordConfirm" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator ID="RequiredFieldValidator4" runat="server"
     ControlToValidate="TextBoxPasswordConfirm" ErrorMessage="密码不能为空！"
     Display="Dynamic">密码不能为空！
</asp:RequiredFieldValidator> <br /><br />
<asp:CompareValidator ID="CompareValidator1" runat="server"
     ControlToCompare="TextBoxPassword"   ControlToValidate="TextBoxPasswordConfirm"
     ErrorMessage="密码不一致，重新输入。" Display="Dynamic">密码不一致，重新输入。
</asp:CompareValidator>
```

(4) 按 Ctrl+F5 组合键运行该页，测试 CompareValidator 控件。

5.5　用户控件

有时可能需要控件具有 ASP.NET 内置服务器控件没有的功能。在这种情况下，用户可以创建自己的控件。有两个选择，可以创建用户控件和自定义控件。

用户控件是能够在其中放置标记和服务器控件的容器。然后，可以将用户控件作为一个单元对待，为其定义属性和方法。

自定义控件是编写的一个类，此类从 Control 或 WebControl 派生。

创建用户控件要比创建自定义控件方便很多，因为可以重用现有的控件。用户控件使创建具有复杂用户界面元素的控件极为方便。

1. 用户控件结构

ASP.NET Web 用户控件与完整的 ASP.NET 网页(.aspx 文件)相似，同时具有用户界面

页和代码页。可以采取与创建 ASP.NET 页相似的方式创建用户控件，然后向其中添加所需的标记和子控件。用户控件可以像页面一样包含对其内容进行操作(包括执行数据绑定等任务)的代码。

2. 用户控件与 ASP.NET 网页有以下区别

(1) 用户控件的文件扩展名为.ascx。

(2) 用户控件中没有@Page 指令，而是包含@Control 指令，该指令对配置及其他属性进行定义。

(3) 用户控件不能作为独立文件运行。而必须像处理任何控件一样，将它们添加到 ASP.NET 页中。

(4) 用户控件中没有 HTML、body 或 form 元素。这些元素必须位于宿主页中。

(5) 可以在用户控件上使用与在 ASP.NET 网页上所用相同的 HTML 元素(HTML、body 或 form 元素除外)和 Web 控件。例如，如果要创建一个将用作工具栏的用户控件，则可以将一系列 Button 服务器控件放在该控件上，并创建这些按钮的事件处理程序。

【例 5-16】演示一个实现微调控件的用户控件。在此微调控件中，用户可以单击【向上】和【向下】按钮以滚动文本框中的一系列选择。

① 启动 VS，打开 WebSite5，新建名为 example5-16.aspx 的网页。

② 在【解决方案资源管理器】中，右击网站名 WebSite5，选择【添加】|【新建项】命令，在弹出的对话框中选择【已安装】模板，使用默认名称为 WebUserControl1.ascx，单击【添加】按钮，用户控件文件 WebUserControl.ascx 就添加到解决方案中了。

③ 单击 WebUserControl1 的【设计】标签，切换到如图 5-22 所示的【设计】视图，然后在【工具箱】中依次双击 TextBox 控件、Button 控件，添加 1 个 TextBox 控件，2 个 Button 控件。TextBox 控件的 ReadOnly 属性设置为 True，ID 属性设置为 TextBoxColor；其中一个 Button 控件的 ID 属性设置为 ButtonUp，Text 属性设置为 Up；另一个 Button 控件的 ID 属性设置为 ButtonDown，Text 属性设置为 Down。

图 5-22　创建数据库文件夹

④ 在WebUserControl1的【设计】视图中双击，切换到WebUserControl1的代码视图，即打开WebUserControl.ascx.cs文件，定义变量，为Page_Load事件、Up按钮和Down按钮的Click事件添加如下代码。

```
public partial class WebUserControl : System.Web.UI.UserControl
{
      protected int currentColorIndex;
      protected string[] colors ={ "Red", "Green", "Blue", "Yellow", };
      protected void Page_Load(object sender, EventArgs e)
      {//IsPostBack 的值指示是正为响应客户端回发而加载用户控件，还是正第一次加载和访问、
       //用户控件。如果是正为响应客户端回发而加载用户控件，则为 true；否则为 false。
            if (IsPostBack)
            {
                  currentColorIndex = Int16.Parse(ViewState["currentColorIndex"].ToString());
            }
```

```
            else
            {
                currentColorIndex = 0;
                DisplayColor();
            }
        }
        private void DisplayColor()
        {
            TextBoxColor.Text = colors[currentColorIndex];
           //在文本框文字改变时字体的颜色也相应改变
           TextBoxColor.ForeColor = System.Drawing.Color.FromName(colors[currentColorIndex]);
           String strColor = colors[currentColorIndex];
           Response.Write("<body    bgColor="+ strColor +"></body>");//让网页背景改变颜色。
           //下面的代码演示如何以多种不同的字号显示相同的 HTML 文本。
           for (int i=0; i<10; i++)
           {
                Response.Write("<font size="+i +"> Hello World! </font>");
           }
           //下面的代码演示以多种不同的颜色显示相同的 HTML 文本。
           String strColor2 = colors[(currentColorIndex+1)%4];
           Response.Write("<font color=" + Color.FromName(strColor2).ToKnownColor () +
                "> 岁月无情增中减，书香有味苦后甜 </font>");
           ViewState["currentColorIndex"] = currentColorIndex.ToString();
        }
        protected void BottonUp_Click(object sender, EventArgs e)
        {
            if (currentColorIndex == 0)
            {
                currentColorIndex = colors.Length - 1;
            }
            else
            {
                currentColorIndex -= 1;
            }
            DisplayColor();
        }
        protected void ButtonDown_Click(object sender, EventArgs e)
        {
            if (currentColorIndex ==(colors .Length -1))
            {
                currentColorIndex = 0;
            }
            else
            {
                currentColorIndex += 1;
            }
            DisplayColor();
        }
    }
```

⑤ 保存用户控件。切换到页面的【设计】视图，从【解决方案资源管理器】中两次拖动用户控件 WebUserControl1 到网页设计视图上，如图 5-23 所示。

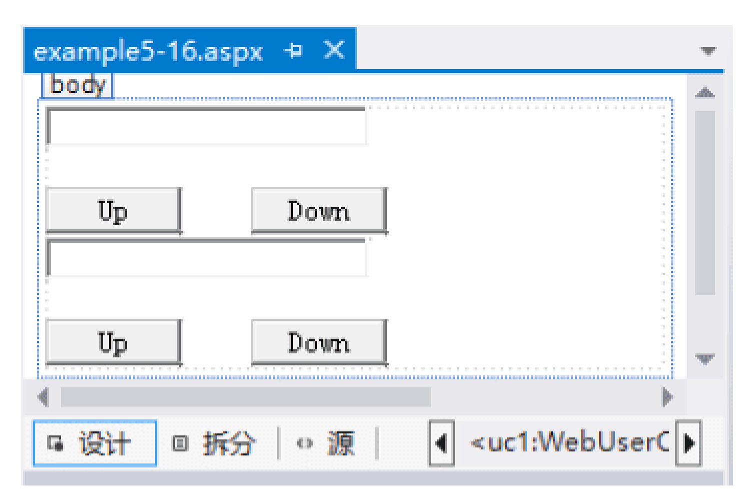


图 5-23　用户控件示例

⑥ 按 Ctrl+F5 组合键运行该页。单击 Up、Down 按钮，分析显示结果。

5.6　本章小结

本章详细介绍了大量的 ASP.NET 服务器控件。由于这些控件如此重要，而且在每个 ASP.NET 应用程序中都会用到，因此，了解工具箱中有哪些控件可用、它们各自的用途、它们的使用方法等非常关键。

本章主要介绍了网页开发中常用到的 HTML 元素和 HTML 服务器控件、标准服务器控件、验证控件、用户自定义控件等内容。这些常用控件基本能够满足 ASP.NET 应用程序的开发。

5.7　思考和练习

1. 在 Asp.net 中所有的自定义用户控件都必须继承自哪个类？
2. 在 ASP.NET 中，类 System.Web.UI.Page 可以被继承么？
3. ASP.NET 框架中，服务器控件是为配合 Web 表单工作而专门设计的。服务器控件有哪两种类型？
4. ASP.NET 中，在 Web 窗体页上注册一个用户控件，指定该控件的名称为 Mike，正确的注册指令是什么？
5. 常见的验证控件有哪些？功能如何？

第6章　页面切换与导航

设计一个网站时，用一个网页完成所有功能是不可能的，通常会按照不同的功能将其划分成各自独立的模块进行处理，所以一个网站通常是由很多网页组成的。这样就需要在不同的网页间进行切换，还可能用到网页间的数据传递或数据共享。

当应用程序或网站越来越复杂时，如何管理成百上千个链接、如何具有清晰的层次结构感就变得越来越重要。为了让登录网站的用户能够容易地知道所处的位置及顺利地访问目的网页，还须为网站添加网站导航。

本章首先介绍几种常用的页面切换方法和数据传递方法，然后介绍与网站导航相关的控件及其基本用法。

本章的学习目标：

- 了解页面切换、数据传递和导航的基础知识。
- 掌握页面之间的切换方法。
- 掌握页面之间的数据传递方法。
- 掌握网站的导航方法。

6.1　页面切换

在 ASP.NET Web 应用程序中，有多种页面切换的方法，常用的有以下几种。

(1) 利用超链接直接跳转切换到其他页面。例如，使用<a>标记或者 HyperLink 控件直接链接到目标页面。

(2) 利用 Button、ImageButton 和 LinkButton 等控件的 PostBackUrl 属性跳转切换到目标页面。

(3) 使用 Response.Redirect 或 Server.Transfer 方法跳转到目标页面。

6.1.1　超链接方式

从一个页面切换到另一个页面最简单的方法就是使用超链接。超链接方法有两种：一种是通过 HTML 的<a>标记实现，另一种是使用 HyperLink 控件实现。

1. 通过<a>标记实现页面切换

通过标记<a>标记的 href 属性，链接到目标页面，示例代码如下。

```
<a href="Page2.aspx">进入页面 2</a>
```

以上代码当用户单击“进入页面 2”链接文字时，跳转切换到 Page2.aspx 页面。

2. 通过 HyperLink 控件实现页面切换

通过 HyperLink 控件的 NavigateUrl 属性，链接到目标页面，示例代码如下。

```
<asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="~/page2.aspx">进入页面 2
</asp:HyperLink>
```

以上代码同样实现了前面标记<a>的功能。

【例 6-1】演示如何利用 HyperLink 控件链接到其他页面。

(1) 运行 VS，新建网站 WebSite6。

(2) 在应用程序中添加一个名为 Page1.aspx 的网页，添加一个名为 Page2.aspx 的网页。

(3) 切换到 Page1.aspx 的【设计】视图，向页面中拖放一个 HyperLink 控件，使用默认的控件名称，然后设置 HyperLink 控件的 NavigateUrl 属性。在 HyperLink 控件的【属性】面板中，单击 NavigateUrl 右侧的按钮，在弹出的【选择 URL】对话框中选择 Page2.aspx，单击【确定】按钮，使其属性值为 Page2.aspx，如图 6-1 所示。并设置 HyperLink 控件的 Text 属性值为“进入页面 2”。

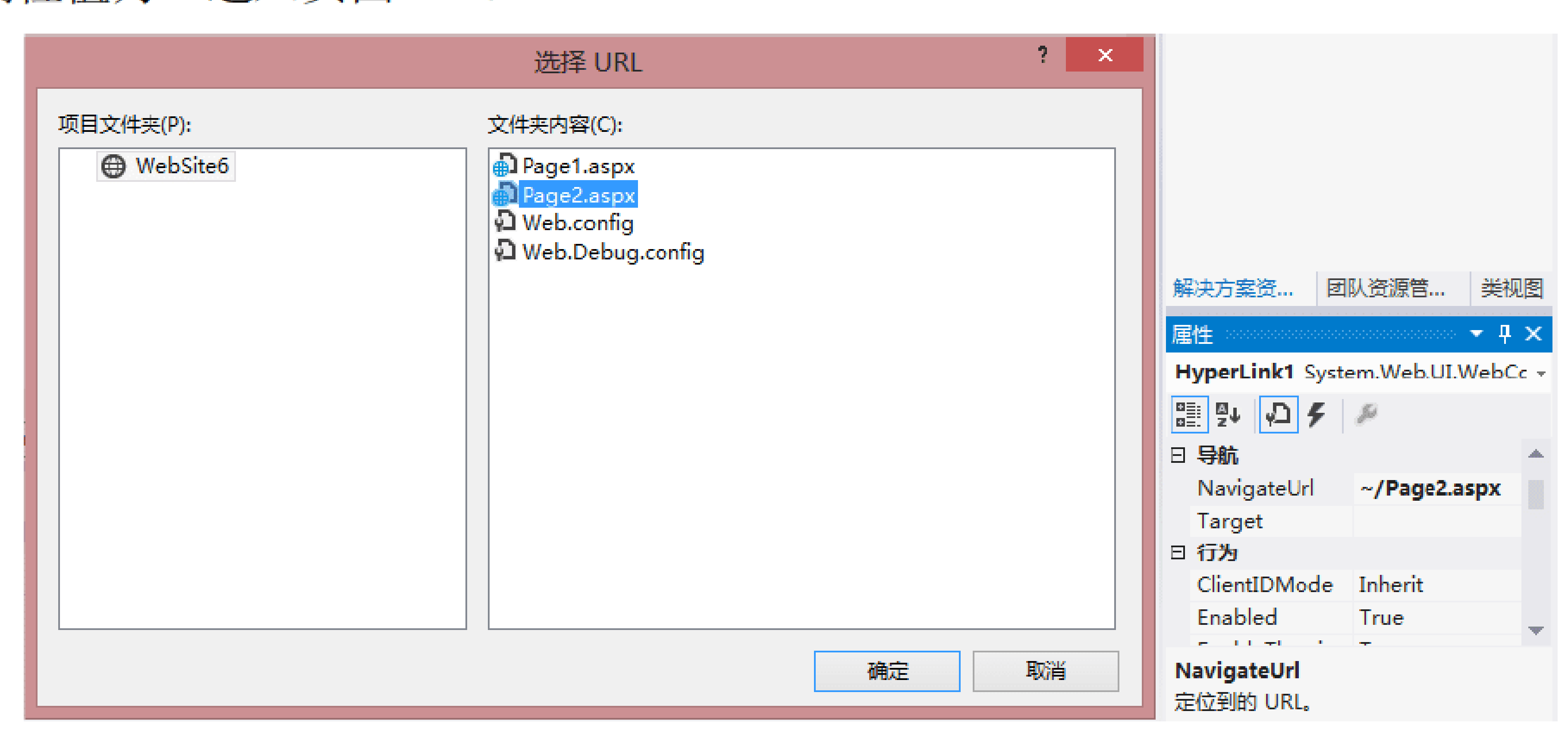


图 6-1　设置 HyperLink 的 NavigateUrl 属性

(4) 切换到 Page1.aspx 网页，按 F5 键调试运行。

HyperLink 控件与<a>标记能够实现相同的链接效果，但是，使用 HyperLink 控件可以在服务器端动态设置 NavigateUrl 属性，示例代码如下。

```
Protected void Page_Load(object sender, EventArgs e)
{
    If  <判断条件>
{
        HyperLink1.NavigateUrl="newPage.aspx";        *判断条件为“真”时，执行此语句
    }
    else
    {
        HyperLink1.NavigateUrl="";                     *判断条件为“假”时，执行此语句
    }
}
```

超链接方式的特点是，当用户单击超链接时，客户端浏览器会直接请求链接的目标网页，因此服务器不会将目标网页与链接前的网页进行关联。也就是说，服务器不会将与源网页有关的任何信息传递到目标网页，除非在目标网页的 URL 上指定查询字符串。

在同一个 Web 应用程序中，如果使用超链接切换到另一个网页，可以通过使用会话状态(Session)来共享源网页和目标网页中的信息。

6.1.2　控件的 PostBackUrl 属性方式

在 Button、LinkButton 和 ImageButton 控件中，有一个 PostBackUrl 属性，可以利用该属性切换到其他页面，这种切换方式称为跨页发送。

跨页发送与超链接都是通过用户操作来切换到其他页面。但是，在跨页发送中，服务器会将源网页上控件的值发送到目标网页。如果源网页和目标网页属于同一个 Web 应用程序，目标网页还可以访问源网页的公共属性。

利用按钮 Button、LinkButton 和 ImageButton 控件的 PostBackUrl 属性切换到其他页面的操作过程与利用超链接切换到其他页面的操作过程相似。

【例 6-2】演示如何利用 Button、LinkButton 和 ImageButton 控件的 PostBackUrl 属性跳转到其他页面。

(1) 运行 VS，打开 WebSite6 网站。

(2) 在应用程序中添加一个名为 PageOne.aspx 的网页，再分别添加名为 PageTwo.aspx、PageThree.aspx、PageFour.aspx 的网页，并分别在 PageTwo.aspx、PageThree.aspx、PageFour.aspx 的网页中添加文字“欢迎来到第二个页面。”、“欢迎来到第三个页面”和“欢迎来到第四个页面”，以便运行时将不同页面区分开。

(3) 切换到 PageOne.aspx 的【设计】视图，向页面中拖放 1 个 Button 控件，1 个 LinkButton 控件和 1 个 ImageButton 控件，均使用默认的控件名称。

分别设置这几个控件的 PostBackUrl 属性。在 Button 控件的【属性】面板中，单击 PostBackUrl 右侧的按钮，在弹出的【选择 URL】对话框中选择 PageTwo.aspx，单击【确定】按钮，使其属性值为 PageTwo.aspx，如图 6-2 所示。并设置 Button 控件的 Text 属性值为“单击 Button 转到第二个页面”。

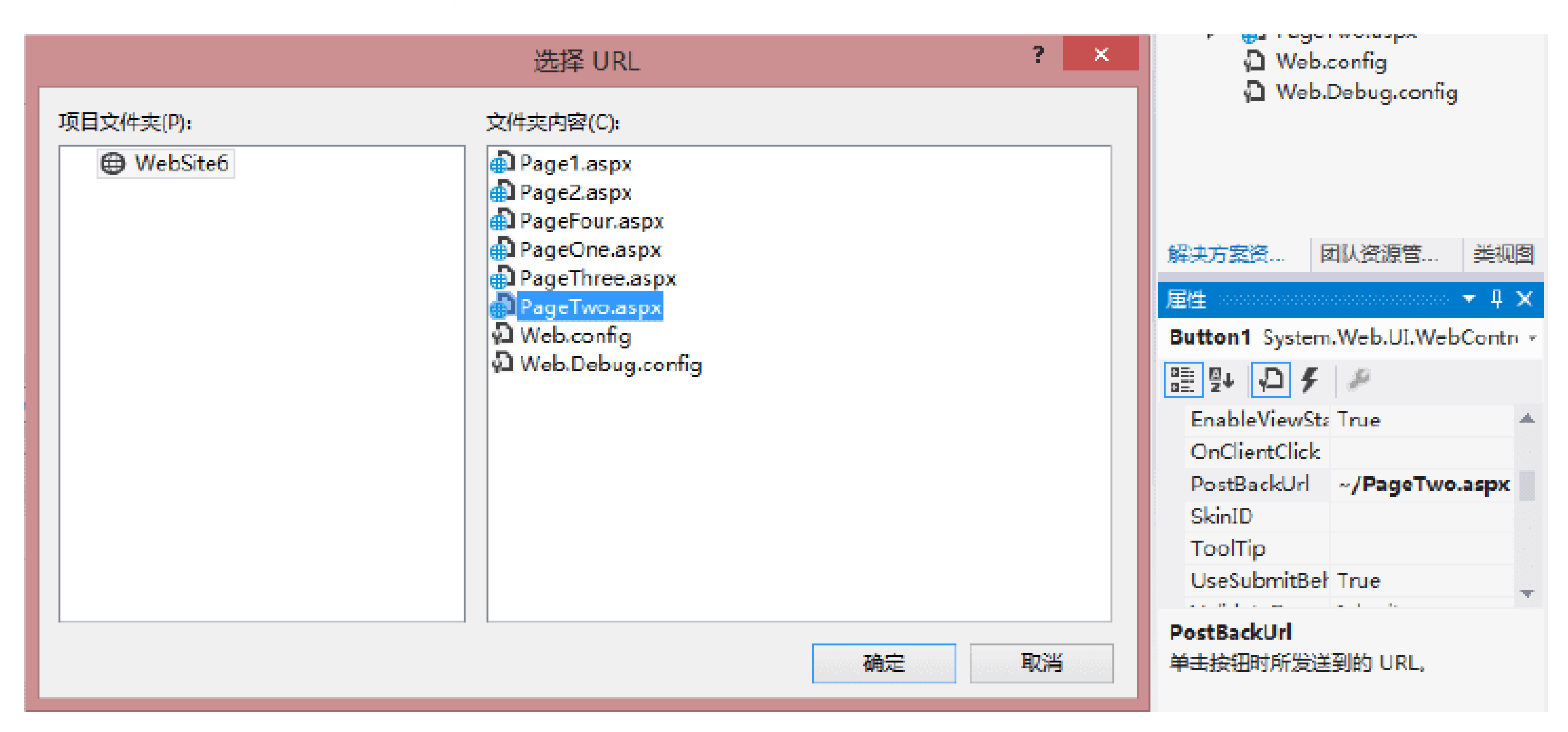


图 6-2　设置 Button 控件的 PostBackUrl 属性

类似地，设置 LinkButton 控件的 PostBackUrl 属性值为 PageThree.aspx，Text 属性值为【单击 LinkButton 转到第三个页面】，设置 ImageButton 控件的 PostBackUrl 属性值为 PageFour.aspx，另外，设置其 ImageUrl 属性。

(4) 切换到 Page1.aspx 网页，按<F5>键调试运行。

6.1.3　Response.Redirect 或 Server.Transfer 方式

前面介绍的超链接和 Button、ImageButton、LinkButton 控件的 PostBackUrl 属性，能够从一个页面跳转到其他页面，但这种页面切换方式完全由用户控制。但是，有时可能需要在程序运行过程中通过代码来控制。例如，在什么情况下需要跳转到另一个页面，这时可以利用Response对象的Redirect方法或者Server对象的Transfer方法来实现页面的动态跳转。

通过 Response 对象的 Redirect 方法与 Server 对象的 Transfer 方法进行页面切换的方式基本相似，但也有些细微区别。

1. Response.Redirect 方法

Response.Redirect 方法会使浏览器链接到一个指定的URL。当服务器调用Response.Redirect方法时，该方法会创建一个应答，应答头中指出目标网页已经改变，同时指出新目标的URL。客户端浏览器从服务器收到该应答，再利用应答头中的信息发出一个对新页面URL的请求。这就是说，使用Response.Redirect方法时，重定向操作发生在客户端，共涉及两次与服务器的通信(两个来回)：第一次是对原始页面的请求，得到一个应答；第二次是请求应答中声明的新页面，得到重定向之后的页面。

2. Server.Transfer 方法

Server.Transfer 方法直接在服务器端把执行流程从当前的页面转到同一个应用程序的另一个页面。调用 Server.Transfer 方法时，当前页面终止执行，然后将执行流程转入另一个页面，但新的页面仍使用前一页面创建的应答流。

用 Server.Transfer 方法从一个页面切换到另一个页面，浏览器中的 URL 不会改变，因为重定向完全在服务器端进行，浏览器根本不知道服务器已经执行了一次页面切换。

Server.Transfer 方法有两种重载的形式，一种是只有一个参数，该参数指出新页面的URL，这种情况下，服务器不会把页面中的表单数据或查询字符串从一个页面传递到另一个页面。这种用法在后面的【例 6-4】中有所体现。另一种重载形式包含两个参数，第一个参数仍是新页面的 URL，而第二个参数则指出是否保留原页面的表单数据和查询字符串以供新页面查询。这种用法在后面的【例 6-3】中有所体现。

3. Response.Redirect 和 Server.Transfer 方法的区别

Response 对象的 Redirect 方法与 Server 对象的 Transfer 方法在进行页面切换时，有以下区别。

(1) Response.Redirect 方法不限于应用程序与.aspx 网页，可以重定向到任何页面，任何文件，如.rar 文件，以及重定向到其他应用程序中的页面。如果重定向的目标不是网页，

浏览器会弹出相应的对话框，用户可以选择是直接打开还是保存到本地磁盘上。Server.Transfer 方法则只能在同一个应用程序的.aspx 网页之间跳转。

(2) 使用 Response.Redirect 方法时，切换到另一个页面之后，浏览器地址栏将显示新页面的 URL。当传递的消息比较敏感时，这种方法会在新页面的 URL 中泄露消息。而 Server.Transfer 方法在跳转到目标页面之后，显示的仍然是原来页面的地址，因此能够很好地保护敏感信息。

6.1.4　页面间的数据传递

在实际应用中，一个页面可能需要访问或者使用另一个页面中的数据，这就需要在不同页面之间传递数据。下面是几种常见数据传递方法。

1. 通过 QueryString 传递数据

通过 QueryString 传递数据时，把要传递的数据参数添加到要跳转到的目标页面的 URL 之后，在要传递的第一个数据参数前用英文格式问号(?)将其和目标页面的名称分隔开，参数之间用&分隔，每个参数的形式为“变量名=值”，示例代码如下。

```
http://www.baidu.com?ID=5&Name=Lily
```

而在另一个页面中使用 Request.QueryString[stringName]接收传递过来的参数。示例代码如下。

```
Label1.Text = "学号：" + Server.UrlDecode(Request.QueryString["ID"]);
Label2.Text = "姓名：" + Server.UrlDecode(Request.QueryString["Name"]);
```

【例 6-3】演示如何通过页面 URL 附带参数传递数据。

(1) 运行 VS，打开网站 WebSite6。

(2) 在应用程序中添加页面 FirstPage.aspx，然后在该页面中添加 2 个 TextBox 控件，1 个 Button 控件，均使用默认的控件名称。分别在 2 个 TextBox 控件前添加文字“学号：”和“姓名：”，设置 Button 控件的 Text 属性值为“提交”。然后为 Button 控件添加 Click 事件处理程序如下。

```
protected void Button1_Click(object sender, EventArgs e)
{
    Server.Transfer(string.Format("SecondPage.aspx?学号={0}&姓名
={1}",Server.UrlEncode(TextBox1.Text), Server.UrlEncode(TextBox2.Text)));
}
```

(3) 在应用程序中添加页面 SecondPage.aspx。在该页面中，添加 2 个 Label 控件，均使用默认名称，然后为该页面添加 Page_load 事件处理程序如下。

```
protected void Page_Load(object sender, EventArgs e)
{
     Label1.Text = "学号：" + Server.UrlDecode(Request.QueryString["学号"]);
     Label2.Text = "姓名：" + Server.UrlDecode(Request.QueryString["姓名"]);
```

```
}
```

(4) 切换到页面 FirstPage.aspx，按 F5 键调试运行，运行效果如图 6-3 和图 6-4 所示。

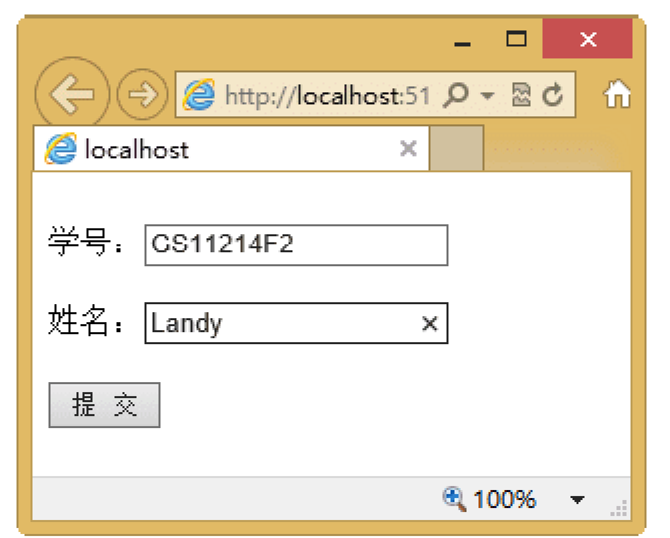
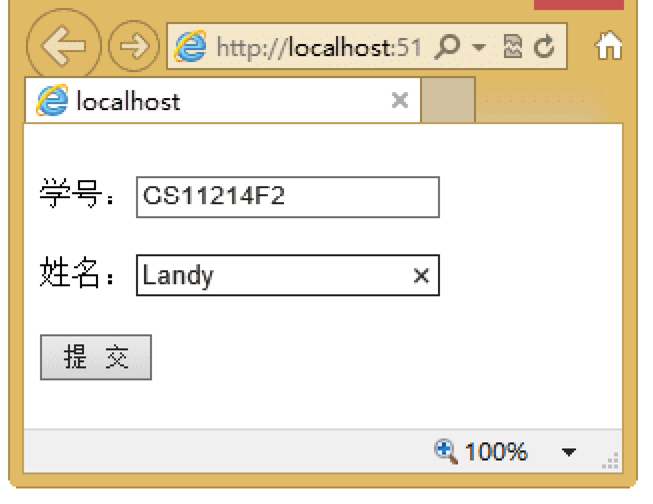



图 6-3　在第一个页面传递数据的效果

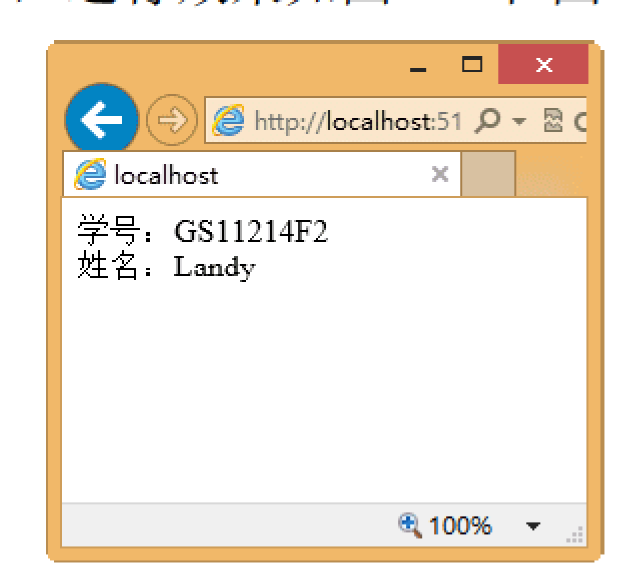


图 6-4　在第二个页面接收数据的效果

本例使用了 QueryString 传递数据，其中使用&作为参数之间分隔符。为了保证接收方能够得到正确的结果，需要在传递数据参数前使用 Server.UrlEncode 方法对字符串进行编码，接收方接收后再使用 Server.UrlDecode 方法对相应的字符串进行解码。UrlEncode 将指定的代码以 URL 格式进行编码。UrlDecode 将 URL 格式代码进行解码。

需要注意的是，如果直接运行 SecondPage.aspx，由于没有接收到任何传递的参数，Request.QueryString[stringName]将返回一个空字符串。

2. 通过 Session 对象传递数据

由于每个用户都可以有自己的 Session，而创建的 Session 对象并不局限于某一个页面，所以可以通过 Session 对象在多个页面之间共享数据。

【例 6-4】演示如何通过 Session 对象传递数据。

(1) 运行 VS，打开 WebSite6。

(2) 在应用程序中添加 1 个名为 Session1.aspx 的网页，在该页面中，添加 2 个 TextBox 控件，1 个 Button 控件，均使用默认的控件名称。分别在 2 个 TextBox 控件前添加文字“学号：”和“姓名：”，设置 Button 控件的 Text 属性值为“提交”。然后为 Button 控件添加 Click 事件处理程序如下。

```
protected void Button1_Click(object sender, EventArgs e)
{
    Session["学号"] = TextBox1.Text;
    Session["姓名"] = TextBox2.Text;
    Server.Transfer("Session2.aspx");
}
```

(3) 在应用程序中添加页面 Session2.aspx，在该页面中，添加 2 个 Label 控件，均使用默认的控件名称，然后为该页面添加 Page_Load 事件处理程序如下。

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Session["学号"] != null)
    {
        Label1.Text = "学号：" + Session["学号"].ToString();
```

```
        }
        if (Session["姓名"] != null)
        {
            Label2.Text = "姓名：" + Session["姓名"].ToString();
        }
    }
```

(4) 切换到 Session1.aspx 网页，按 F5 键调试运行。运行效果与【例 6-3】相似。

需要注意的是，通过这种方法传递数据会占用服务器的内存资源，并且每个用户的 Session 信息将一直存储在内存中，直到 Session 过期，因此系统开销较大。

下面介绍通过 PreviousPage 属性获取上一个页面数据的方法，这个方法适用于在页面间传递较多的信息。

3. 通过 PreviousPage 属性获取数据

如果两个页面属于同一个应用程序，当利用 Server.transfer 方法或者利用按钮控件的 PostBackUrl 属性在这两个页面间切换时，目标页面可以使用 PreviousPage 属性来获取前一个页面中的控件值或公共属性。

如果要获取第一个页面中的公共属性，则需要在目标页面的源代码中添加如下代码。

```
<%@ PreviousPageType VirtualPath="~/SourcePage.aspx" %>
```

但是，如果目标页面不需要获取前一个页面的公共属性，而仅仅需要获取前一个页面中的控件值，则不需要添加这行代码。

【例 6-5】演示如何利用 PreviousPage 属性获取前一个页面的控件值。

(1) 运行 VS，打开 WebSite6。

(2) 在应用程序中添加页面 example6-5-1.aspx，在该页面中，添加 2 个 TextBox 控件，1 个 Button 控件，均使用默认的控件名称。 分别在 2 个 TextBox 控件前添加文字“学号：”和“姓名：”，设置 Button 控件的 Text 属性值为“提交”。然后为 Button 控件添加 Click 事件处理程序如下。

```
protected void Button1_Click(object sender, EventArgs e)
{
    Server.Transfer("example6-5-2.aspx");
}
```

(3) 在应用程序中添加页面 example6-5-2.aspx，在该页面中，添加 2 个 Label 控件，均使用默认的控件名称，然后为该页面添加 Page_load 事件处理程序如下。

```
protected void Page_Load(object sender, EventArgs e)
{
    if (PreviousPage != null)
    {
        TextBox Textbox1 = (TextBox)PreviousPage.FindControl("TextBox1");
        TextBox Textbox2 = (TextBox)PreviousPage.FindControl("TextBox2");
        Label1.Text = "学号：" + Textbox1.Text;
```

```
            Label2.Text = "姓名：" + Textbox2.Text;
        }
    }
```

(4) 切换到 example6-5-1.aspx 网页，按 F5 键调试运行，运行效果与【例 6-3】相似。

在 example6-5-2.aspx 网页中，使用了 PreviousPage.FindControl 获取 example6-5-1.aspx 页面中的控件值。这种方法与利用 Session 对象传递数据相比，不需要一直占用服务器资源，适合于页面间传递较多的信息时使用，使页面间的数据传递更加灵活方便。

【例 6-5】演示了如何通过 PreviousPage 属性获取源页面的控件值。如果要在 example6-5-2.aspx 网页中访问 example6-5-1.aspx 网页中的公共属性，可以按照以下步骤进行。

(1) 在 example6-5-1.aspx 中定义属性。代码如下。

```
public string Number
{
    get {return TextBox1.Text;}
}
public string UserName
{
    get {return TextBox2.Text;}
}
protected void Button1_Click(object sender, EventArgs e)
{
    Server.Transfer("example6-5-2.aspx");
}
```

(2) 在 example6-5-2.aspx 的【源】视图中添加以下代码。

```
<%@PreviousPageType VirtualPath="~/example6-5-1.aspx"%>
```

然后就可以在example6-5-2.aspx的后台代码中使用example6-5-1.aspx中定义的Number和UserName属性了。代码如下。

```
Protected void Page_load(object sender, EventArgs e)
{
    If(PreviousPage!=null)
    {
        Label1.Text="学号："+PreviousPage.Number;
        Label2.Text="姓名："+PreviousPage.UserName;
    }
}
```

6.2　网站导航

当应用程序或网站越来越复杂时，如何管理成百上千个链接、如何让用户具有清晰的层次结构感就变得越来越重要了。ASP.NET 提供了一种称为站点导航模型的导航功能集合，使开发人员在创建站点导航功能时更容易。该功能能够为用户提供一致的站点导航方

式。本节主要来介绍网站导航功能。

6.2.1　ASP.NET 网站导航功能概述

网站导航主要提供了如下功能。

(1) 使用站点地图描述网站的逻辑结构。添加或移除页面时，开发人员可以简单地通过修改站点地图来管理页面导航。

(2) 提供导航控件，在页面上显示导航菜单。导航菜单以站点地图为基础。

(3) 可以以代码方式使用 ASP.NET 网站导航，以创建自定义导航控件或修改在导航菜单中显示的信息的位置。

ASP.NET 4.5 提供了 3 种导航 Web 控件。

(1) SiteMapPath：这个 Web 控件提供一个面包条，它是一行文本，显示用户当前在网站结构中的位置。例如，在网上书店中，如果用户浏览到《Visual Basic.NET》，面包条可能类似于“主页→计算机→编程类→Visual Basic.NET”，其中每部分(如主页、计算机等)都显示为返回到前一部分的链接。面包条能够让用户快速地查看在网站中的当前位置，并沿逻辑层次结构向上导航。如图 6-5 所示显示了运行中的 SiteMapPath 控件。

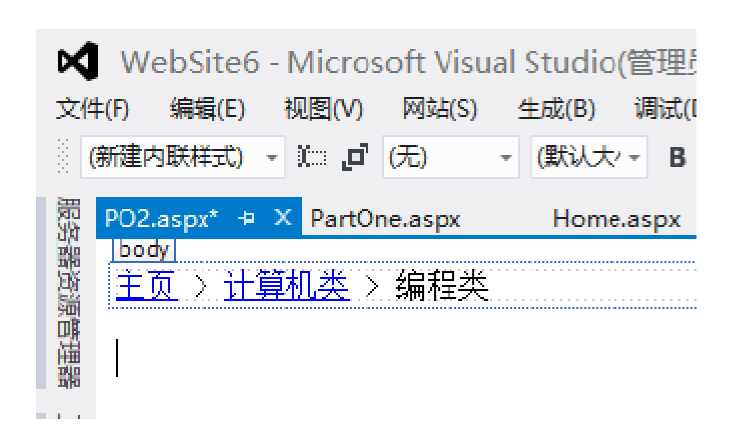


图 6-5　SiteMapPath 控件。

(2) Menu：这个 Web 控件提供网站结构的层次视图。对于学校的网站，顶层菜单将包含主类别(如学校介绍、机构设置、新闻等)，每个菜单项又可以有各自的子菜单，显示各自的子类别。关于运行中的 Menu 控件可参见如图 6-6 所示。

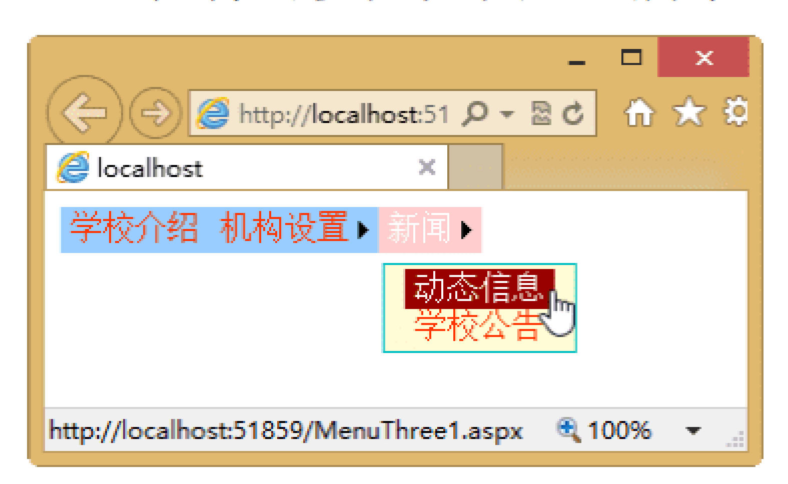


图 6-6　Menu 控件

(3) TreeView：树视图提供了与菜单相同的数据，唯一的区别是显示数据的方式。树视图显示的是可展开或可折叠的树，而菜单(Menu)由菜单项和子菜单组成。如图 6-7 所示显示了通过浏览器查看时的 TreeView 控件。

一般情况下，开发人员利用站点地图和 SiteMapPath 控件实现自动导航，利用 Menu 控件或者 TreeView 控件实现自定义导航。

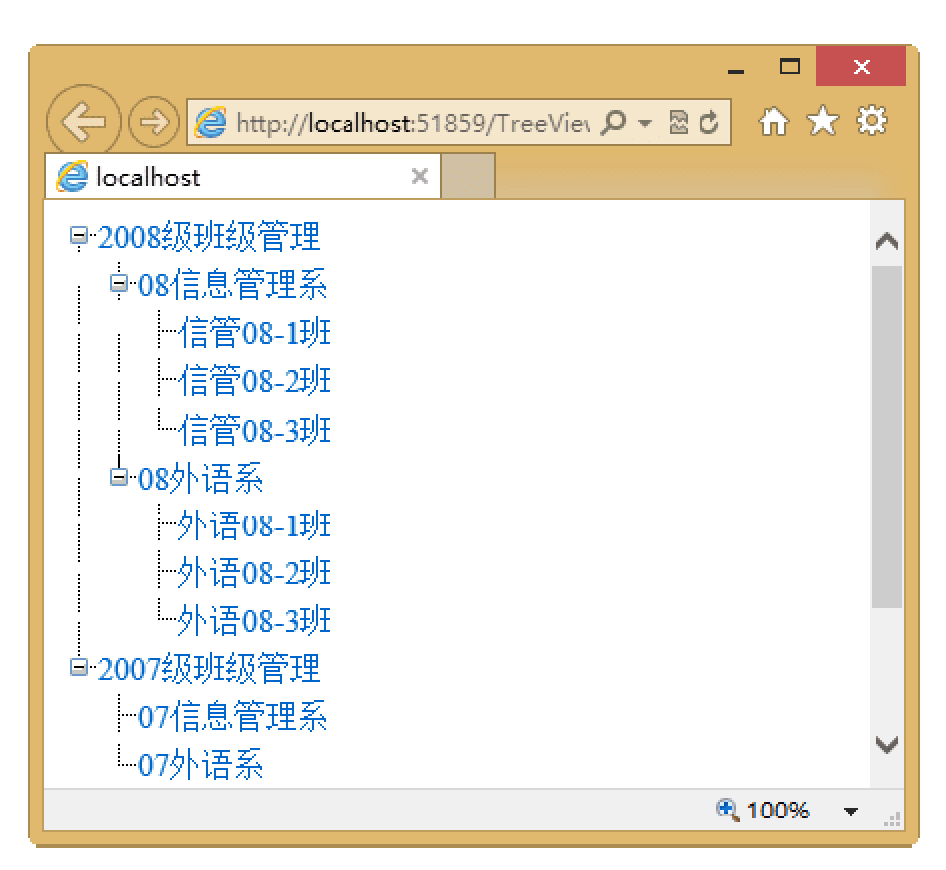


图 6-7　TreeView 控件。

6.2.2　利用站点地图和 SiteMapPath 控件实现网站导航

1. 站点地图概述

站点地图是指网站的整体结构。当开发人员在规划站点导航模型时，应该先规划整个网站的结构。网站的结构包括页面的层次结构、页与页之间的链接位置。例如，一家在线购物网站中可能具有 14 个页面，这些页面可能会具有如下层次结构。

```
主页
 产品
      硬件
           电脑整机
           电脑主板
           监控系统
      软件
           操作系统
           数据库
           ERP 系统
      服务
           培训
           咨询&顾问
           服务支持
```

在创建一个站点导航模型时，开发人员首先应该这样规划站点的逻辑结构，这个结构在未来可能会根据需要添加更多的页面或移除页面。ASP.NET 提供了相关的控件和模板，允许开发人员静态或动态地创建网站的站点地图。

2. 定义站点地图文件

在 ASP.NET 中，为了创建站点地图，最简单的方法是创建一个名为 Web.sitemap 的站点地图文件，下面将面前介绍的站点层次结构用站点地图文件表示出来，操作步骤如下。

(1) 打开网站 WebSite6，在解决方案资源管理器中右击网站名称，在弹出的快捷菜单中选择【添加新项】命令，打开【添加新项】对话框，选择【站点地图】选项，保持默认

名称不变，如图 6-8 所示。单击【确定】按钮，VS 2012 将自动创建一个站点地图文件，该文件保留了基本的站点地图结构，代码如下。

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="" title=""   description="">
        <siteMapNode url="" title=""   description="" />
        <siteMapNode url="" title=""   description="" />
    </siteMapNode>
</siteMap>
```

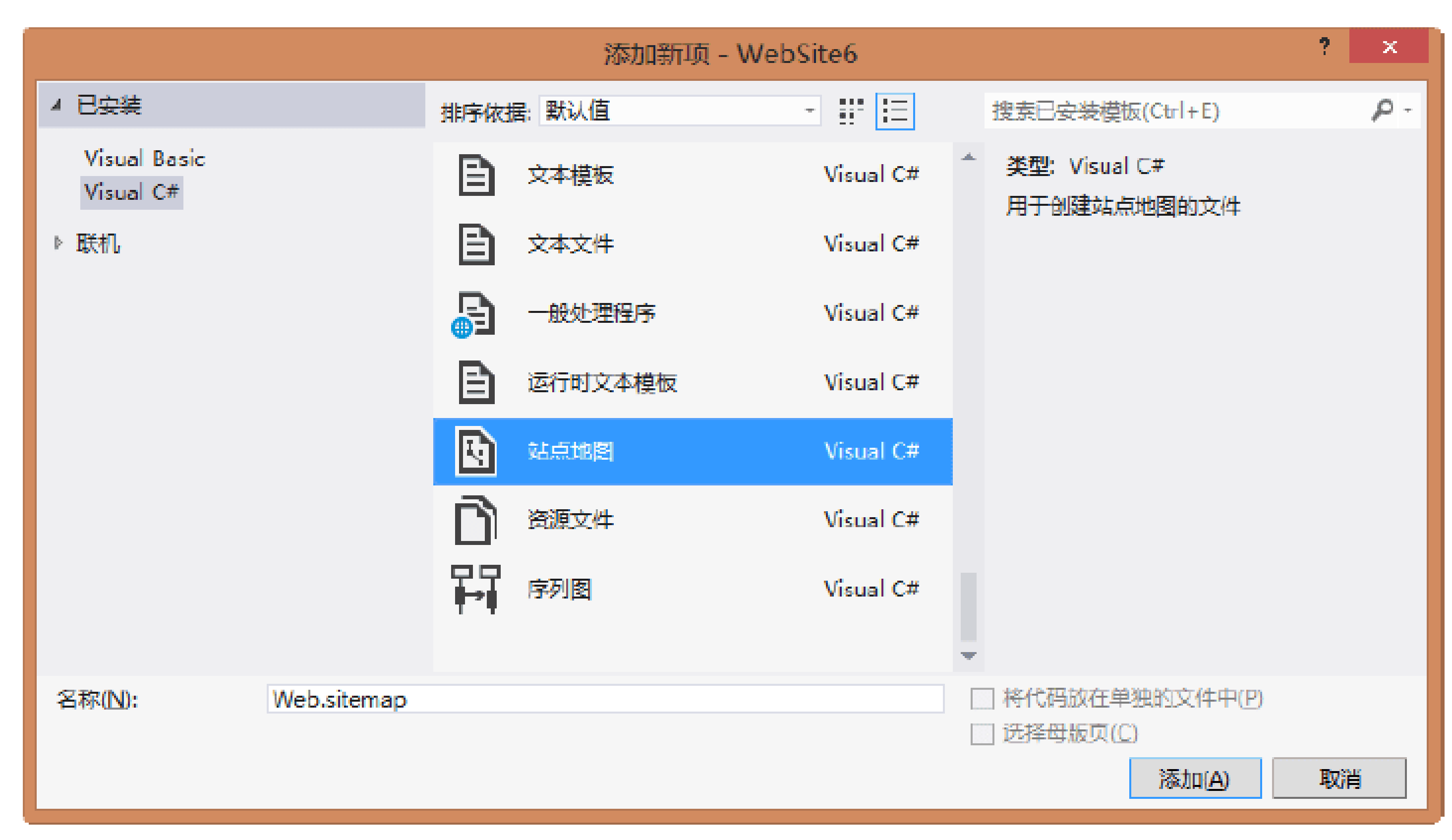


图 6-8　创建站点地图文件

站点地图文件就是一个 XML 文件，该文件以<siteMap>作为根元素，在<siteMap>内至少嵌套一个<siteMapNode>子元素，每个子元素包含如下 3 个主要的属性。

- title：所关联节点的标题文本，这个文本将显示在页面上。
- description：与所关联节点相对应的描述性文本。
- rul:指向一个页面或其他资源的链接地址。

注意：

url 节点并不是必需的。例如，产品和服务节点或许没有链接页面，因此可以不用指定 URL 属性。

3. SiteMapPath 控件的使用

要使用 SiteMapPath 导航控件，首先需要使用站点地图定义网站的结构，创建站点地图文件，然后使用 SiteMapPath 控件实现网站导航。

定义好站点地图以后，就可以使用 SiteMapPath 控件显示导航路径，也就是显示当前页面在网站中的位置。只需要将该控件拖动到站点地图中包含的.aspx 页面上，它就会自动实现导航，不需要开发者编写任何代码。

注意：

只有包含在站点地图中的网页才能被 SiteMapPath 控件导航；如果将 SiteMapPath 控件

放置在站点地图中未列出的网页中，该控件将不会显示任何信息。

SiteMapPath 控件像大多数 Web 控件一样，也有很多可用于定制其外观的属性。表 6-1 所示为 SiteMapPath 控件的常用属性。

表 6-1　SiteMapPath 控件的常用属性

属　性　名	说　　明
CurrentNodeStyle	定义当前节点的样式，包括字体、颜色、样式等
NodeStyle	定义导航路径上所有节点的样式
ParentLevelsDisplayed	指定在导航路径上显示的相对于当前节点的父节点层数。默认值为-1，表示父级别数没有限制
PathDirection	指定导航路径上各节点的显示顺序。默认值为 RootToCurrent，即按从左到右的顺序显示从根节点到当前节点的路径。另一选项为 CurrentToRoot，即按相反的顺序显示导航路径
PathSeparator	指定导航路径中节点之间分隔符。默认值为>，也可自定义为其他符号
PathSeparatorStyle	定义分隔符的样式
RenderCurrentNodeAsLink	是否将导航路径上当前页名称显示为超链接。默认值为 false
RootNodeStyle	定义根节点的样式
ShowToolTips	当鼠标悬停于导航路径的某个节点时，是否显示相应的工具提示信息。默认值为 true，即当鼠标悬停于某节点上时，显示该节点在站点地图中定义的 Description 属性值

下面通过具体例子演示如何利用站点地图和 SiteMapPath 控件实现自动导航。

【例 6-6】创建如图 6-9 所示的站点地图，然后利用 SiteMapPath 控件实现自动导航。

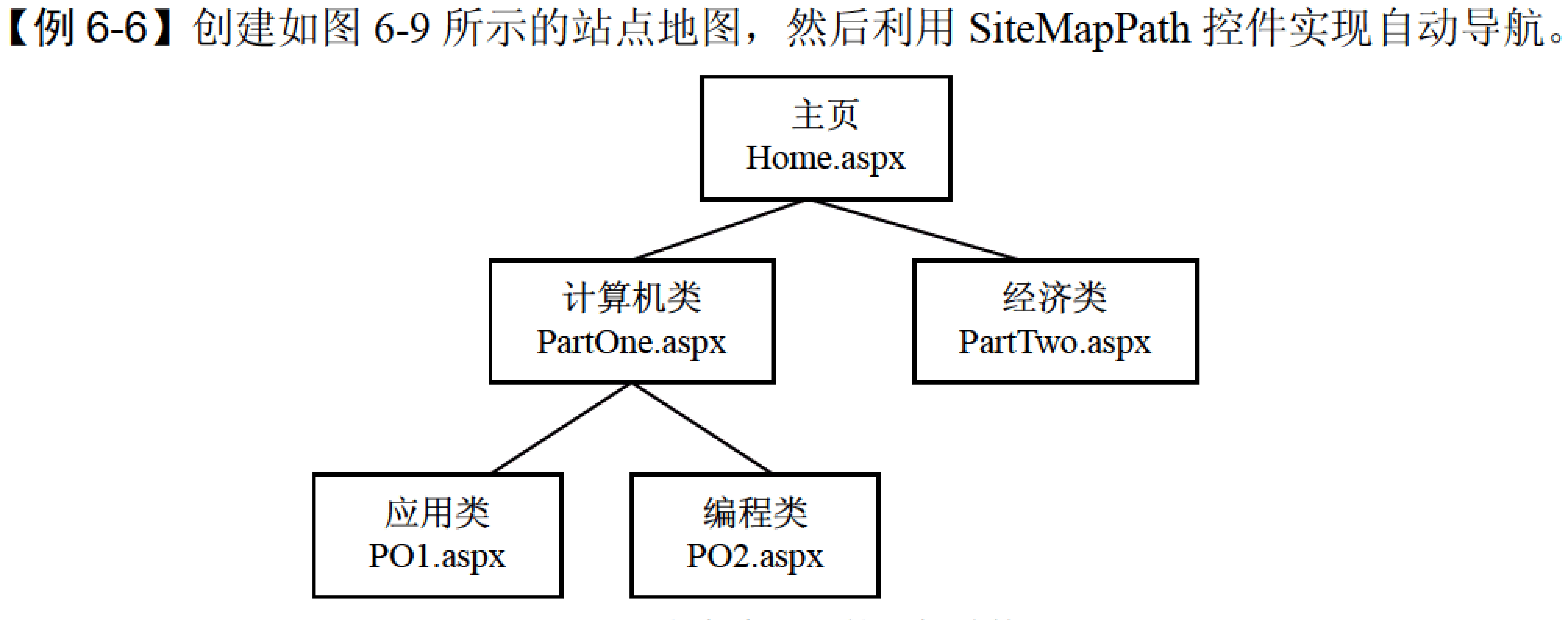


图 6-9　网上书店网站的逻辑结构

(1) 运行 VS，打开 WebSite6。

(2) 在应用程序中新建一个站点地图 Web.sitemap，如图 6-10 所示。

(3) 修改 Web.sitemap 文件的内容如下。

```
<?xml version="1.0" encoding="utf-8" ?>
```

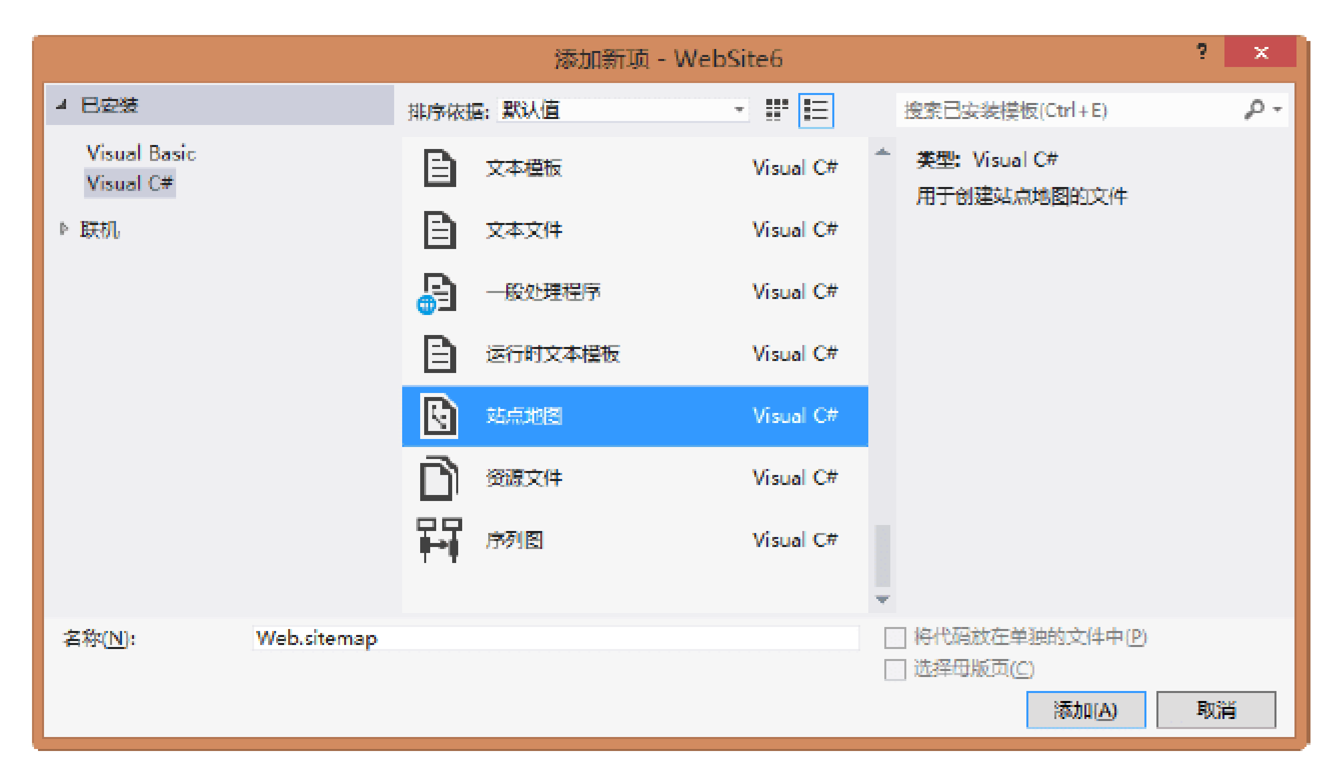


图 6-10　创建站点地图

```
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
 <siteMapNode url="~/Home.aspx" title="主页"  description="Home">
<siteMapNode url="~/PartOne.aspx" title="计算机类"  description="单击此链接转到计算机类" >
    <siteMapNode url="~/PO1.aspx" title="应用类"  description="单击此链接转到应用类" />
    <siteMapNode url="~/PO2.aspx" title="编程类"  description="单击此链接转到编程类" />
    </siteMapNode>
    <siteMapNode url="~/PartTwo.aspx" title="经济类"  description="单击此链接转到经济类" >
    </siteMapNode>
  </siteMapNode>
</siteMap>
```

注意：

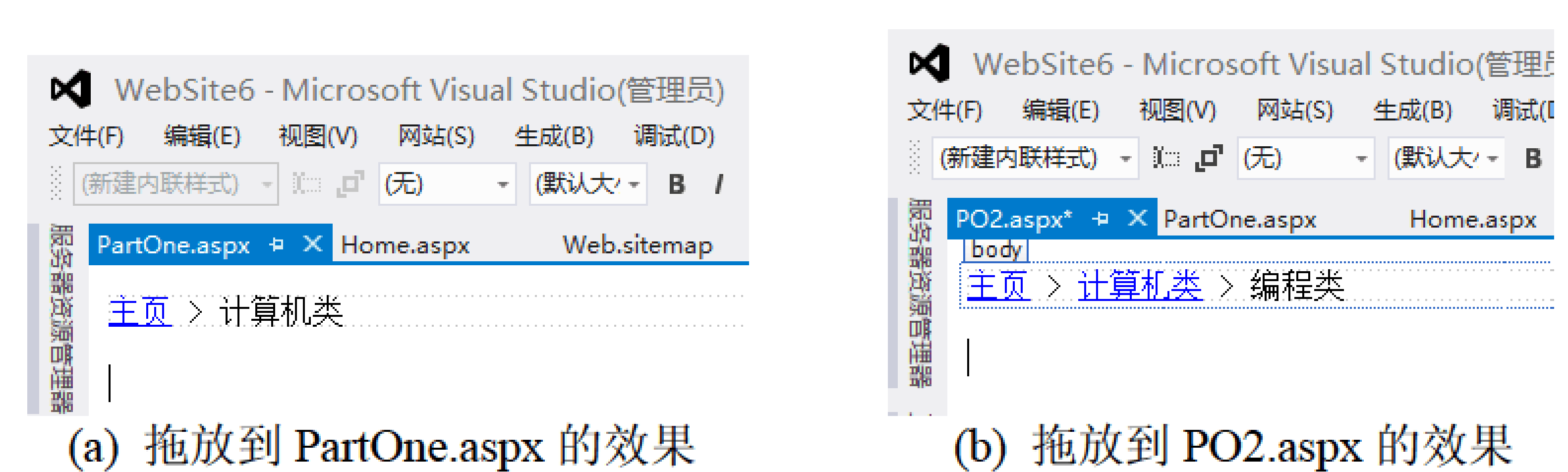
站点地图文件中只能有一个根节点，即位于<sitemap>下方的第一个<siteMapNode>元素中的 Home.aspx 页面。在根节点下可以嵌套任意多个子节点，子节点仍然用<siteMapNode>定义。

(4) 保存文件，完成站点地图设计。定义站点地图之后，即可在导航控件中轻松实现导航功能。

(5) 在【解决方案资源管理器】中，分别添加页面Home.aspx、PartOne.aspx、PartTwo.aspx、PO1.aspx和PO2.aspx。

(6) 切换到 PartOne.aspx 的【设计】视图，向页面拖放 1 个 SiteMapPath 控件，即可看到该页面对应 Home.aspx 的导航路径，如图 6-11(a)所示。

(7) 切换到 PO2.aspx 的【设计】视图，向页面中拖放 1 个 SiteMapPath 控件，即可看到该页面对应 Home.aspx 和 PartOne.aspx 的导航路径，如图 6-11 (b)所示。



(a) 拖放到 PartOne.aspx 的效果　　(b) 拖放到 PO2.aspx 的效果

图 6-11　将 SiteMapPath 控件拖放到页面后的效果

可见，通过站点地图和 SiteMapPath 控件实现自动导航非常方便。如果不希望采用这

种方式导航，也可以利用 Menu 或 TreeView 控件实现自定义导航功能。

6.2.3　通过 Menu 控件实现导航

Menu 控件主要用于创建一个菜单，使用户能够快速选择不同的页面，从而完成导航功能。该控件可以包含一个主菜单和多个子菜单。菜单有静态和动态两种显示模式。静态显示模式是指定义的菜单始终完全显示，动态显示模式指需要用户将鼠标停留在菜单项上时才显示子菜单。

Menu 控件的常用属性如表 6-2 所示。

表 6-2　Menu 控件的常用属性

属　性　名	说　　明
DynamicEnableDefaultPopOutImage StaticEnableDefaultPopOutImage	是否在菜单各项之间显示分隔图像，默认值为 true
DynamicPopOutImageUrl StaticPopOutImageUrl	设置菜单中自定义分隔图像的 URL
DynamicBottomSeparatorImageUrl StaticBottomSeparatorImageUrl	指定在菜单项下方显示图像的 URL。默认值为空字符串("")，即菜单项下方不显示任何图像
DynamicTopSeparatorImageUrl StaticTopSeparatorImageUrl	指定在菜单项上方显示图像的 URL。默认值为空字符串("")，即菜单项上方不显示任何图像
DynamicHorizontalOffset StaticHorizontalOffset	指定菜单相对于其父菜单的水平距离，单位是像素，默认值为 0。该属性值可正可负，为负值时，各菜单之间的距离会缩小
DynamicVerticalOffset StaticVerticalOffset	指定菜单相对于其父菜单项的垂直距离
MaximumDynamicDisplayLevels	设置动态菜单的最大层数，默认值为 3
Orientation	设置菜单的展开方向。有 Horizontal 和 Vertical 两个选项，默认值为 Vertical，即垂直方向

Menu 控件的用法非常灵活，设计者可以利用它定义各种菜单样式，实现类似于 Windows 窗口菜单的功能。

下面通过一个具体的例子演示如何利用 Menu 控件实现自定义导航。

【例 6-7】假定网站的结构如图 6-12 所示，然后利用 Menu 控件在网页中添加一个菜单，实现自定义导航功能。

具体操作步骤如下。

(1) 运行 VS，打开 WebSite6。

(2) 在应用程序中分别添加页面 MenuOne.aspx、MenuTwo1.aspx、MenuTwo2.aspx、MenuTwo3.aspx、MenuThree1.aspx 和 MenuThree2.aspx。

(3) 在应用程序中添加一个页面 MenuExample.aspx，然后切换到【设计】视图，向页面拖放一个 Menu 控件。

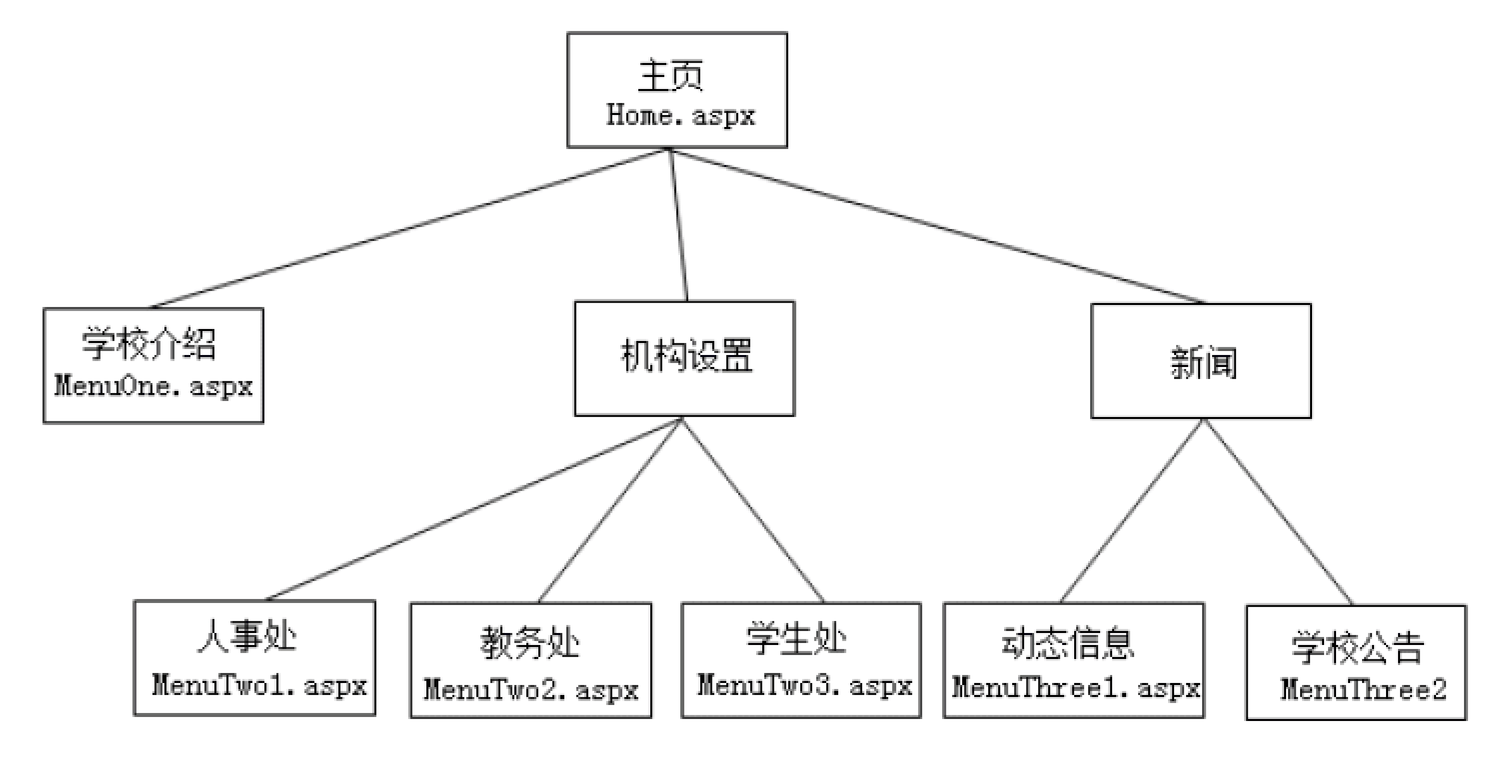


图 6-12　学校网站的逻辑结构

(4) 将 Menu 控件的 Orientation 属性设置为 Horizontal，以便使其横向排列。

(5) 单击 Menu 控件右上方的小三角符号，选择【编辑菜单项】，在弹出的【菜单项编辑器】对话框中，输入各级菜单项，如图 6-13 所示。

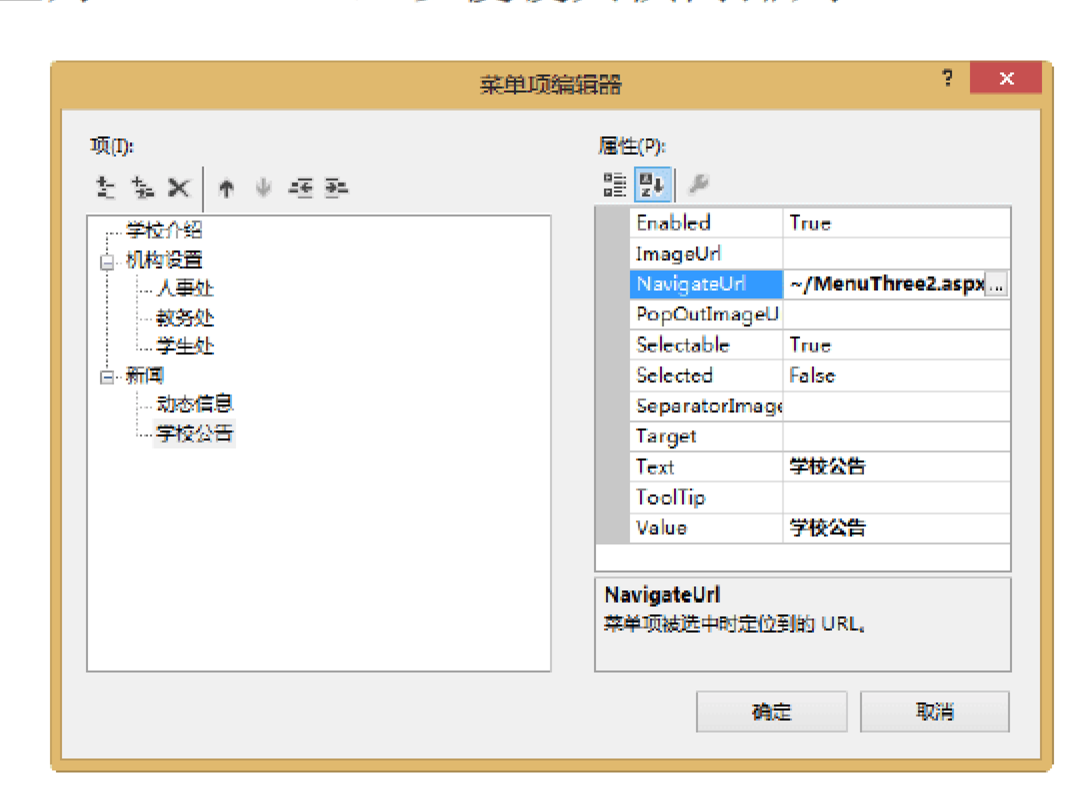


图 6-13　在【菜单编辑器】中编辑菜单

(6) 在【菜单编辑器】窗口右侧的属性选项中，利用 NavigateUrl 属性设置各菜单项链接的网页，全部设置完成后，单击【确定】按钮。

(7) 切换到 MenuExample.aspx 的【源】视图，将<body>和</body>之间的部分改为如下内容。

```
<asp:Menu ID="Menu1" runat="server"
            EnableViewState="False"
            DynamicHorizontalOffset="2"
            DynamicVerticalOffset="5"
            Target="_blank"
            Font-Names="Verdana"
            Font-Size="Medium"
            ForeColor="#FF3300"
            BackColor="#99CCFF"
            StaticSubMenuIndent="10px" Orientation="Horizontal" >
           <StaticHoverStyle BackColor="#FFCCCC" ForeColor="white" />
           <StaticSelectedStyle    BackColor="#FFCC66"/>
           <StaticMenuItemStyle HorizontalPadding="5px" VerticalPadding ="2px" />
           <DynamicHoverStyle BackColor ="#990000" ForeColor ="White" />
           <DynamicMenuStyle BackColor ="#FFFBD6"    BorderColor ="#00C0C0"
            BorderStyle ="Solid" BorderWidth ="1px"
            HorizontalPadding ="10px" VerticalPadding ="2px" />
           <DynamicSelectedStyle BackColor ="#FFCC66" />
           <DynamicMenuItemStyle HorizontalPadding ="5px" ItemSpacing= "2px" />
            <Items>
              <asp:MenuItem Text="学校介绍" NavigateUrl="~/MenuOne.aspx" >
```

```
                    </asp:MenuItem>
                    <asp:MenuItem Text="机构设置" Value="机构设置" >
                        <asp:MenuItem Text="人事处" NavigateUrl="~/MenuTwo1.aspx" Value="人事
处" ></asp:MenuItem>
                        <asp:MenuItem Text="教务处" NavigateUrl="~/MenuTwo2.aspx" Value="教务
处" ></asp:MenuItem>
                        <asp:MenuItem Text="学生处" NavigateUrl="~/MenuTwo3.aspx" Value="学生
处"></asp:MenuItem>
                    </asp:MenuItem>
                    <asp:MenuItem Text="新闻" Value="新闻" >
                        <asp:MenuItem Text="动态信息" NavigateUrl="~/MenuThree1.aspx"
Value="动态信息" ></asp:MenuItem>
                        <asp:MenuItem Text="学校公告" NavigateUrl="~/MenuThree2.aspx"
Value="学校公告" ></asp:MenuItem>
                    </asp:MenuItem>
                </Items>
            </asp:Menu>
```

当然，也可以在【设计】视图下设置 Menu 控件的各种属性得到上面的代码。

(8) 为了便于区别本例子中的各个网页，分别在 MenuOne.aspx、MenuTwo1.aspx、MenuTwo2.aspx、MenuTwo3.aspx、MenuThree1.aspx 和 MenuThree2.aspx 网页中添加文字“欢迎了解学校，以下将对学校进行介绍”、“欢迎来到人事处网页！”、“欢迎来到教务处网页！”、“欢迎来到学生处网页！”、“欢迎了解动态信息！”和“欢迎了解学校公告！”。

(9) 切换到 MenuExample.aspx 网页，按 F5 键调试运行，效果如图 6-14 所示。

图 6-14　Menu 控件的运行效果

6.2.4　通过 TreeView 控件实现导航

TreeView 控件与 Menu 控件相似，都提供了导航功能。TreeView 控件与 Menu 控件的区别是它不再像 Menu 控件由菜单项和子菜单组成，而是用一个可折叠树显示网站的各个部分。根节点下可以包含多个子节点，子节点下又可以包含子节点，最下层是叶节点。访问者可以快速看到网站的所有部分及位于网站结构层次中的位置。树中的每个节点都显示为一个超链接，被单击时把用户引导到相应的部分。

TreeView 控件也包含很多属性，其中常用属性如表 6-3 所示。

表 6-3　TreeView 控件的常用属性

属　性　名	说　　明
CollapseImageUrl	节点折叠后显示的图像。默认情况下，常用带方框的 + 号作为可展开指示图像
ExpandImageUrl	节点展开后显示的图像。默认情况下，常用带方框的 - 号作为可折叠指示图像
EnableClientScript	是否可以在客户端处理节点的展开和折叠事件。默认值为 true
ExpandDepth	第一次显示 TreeView 控件时，树的展开层次数。默认值为 FullyExpand(即-1)，表示全部展开所有节点
Nodes	设置 TreeView 控件的各级节点及其属性

(续表)

属　性　名	说　　明
ShowExpandCollapse	是否显示折叠、展开图像。默认值为 true
ShowLines	是否显示连接子节点和父节点之间的连线。默认值为 false
ShowCheckBoxes	指示在哪些类型节点的文本前显示复选框。共有 5 个属性值：None(所有节点均不显示)、Root(仅在根节点前显示)、Parent(仅在父节点前显示)、Leaf(仅在叶节点前显示)和 All(所有节点前均显示)。默认值为 None

除了表 6-3 所示的 TreeView 的常用属性外，TreeView 控件还有很多与外观相关的属性，可以用来定制 TreeView 的外观。TreeView 控件的外观属性如表 6-4 所示。

表 6-4　TreeView 控件的外观属性

属　性　名	说　　明
HoverNodeStyle	当鼠标悬停于节点上时，节点的样式
LeafNodeStyle	叶节点的样式
LevelStyle	特殊深度节点的样式
NodeStyle	所有节点的默认样式
ParentNodeStyle	父节点的样式
RootNodeStyle	根节点的样式
SelectedNodeStyle	选定节点的样式

下面通过一个例子演示如何利用 Menu 控件实现自定义导航。

【例 6-8】利用 TreeView 控件实现如图 6-15 所示的导航功能，当单击【节点】时，导航到对应的网页。

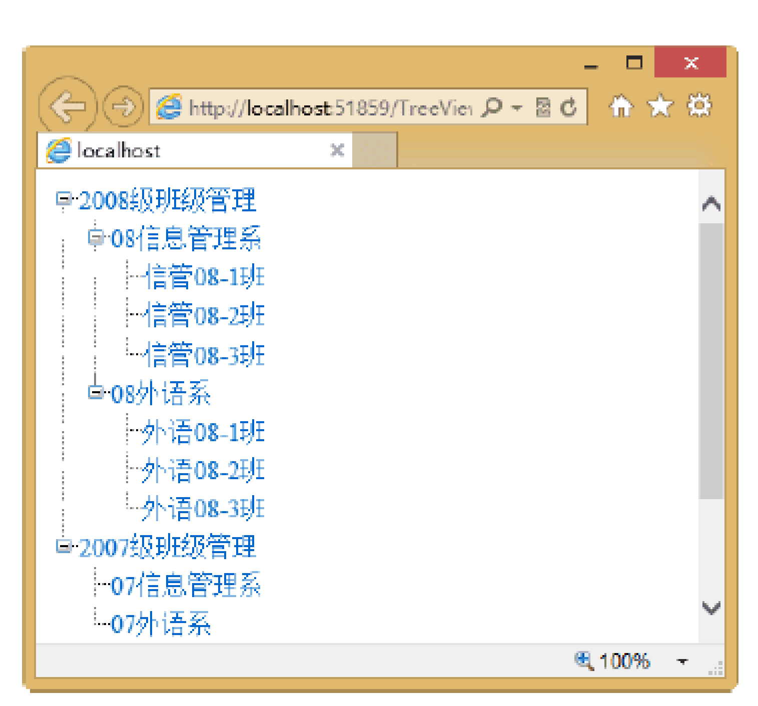


图 6-15　TreeView 导航示例

(1) 运行 VS，打开网站 WebSite6。

(2) 在 WebSite6 中添加 InformationManage_ class1.aspx、InformationManage_class2.aspx、InformationManage_class3.aspx、ForeignLanguage_ class1.aspx、ForeignLanguage_class2.aspx 和 ForeignLanguage_class3.aspx 页面。

(3) 然后添加页面 TreeViewExample.aspx，切换到【设计】视图，拖放一个 TreeView 控件。

(4) 将 TreeView 控件的样式设置为如图 6-16 左侧的样式。

(5) 单击 TreeView 控件右上方的小三角符号，选择【编辑节点】，在弹出的【Treeview 节点编辑器】对话框中，输入各节点名称，如图 6-16 所示。

说明：

为了让读者能看到添加节点后的效果，图 6-16 采用的是添加后重新进入编辑状态看到的效果。如果是第一次添加节点，不会看到图中左侧 TreeView 控件显示的效果。

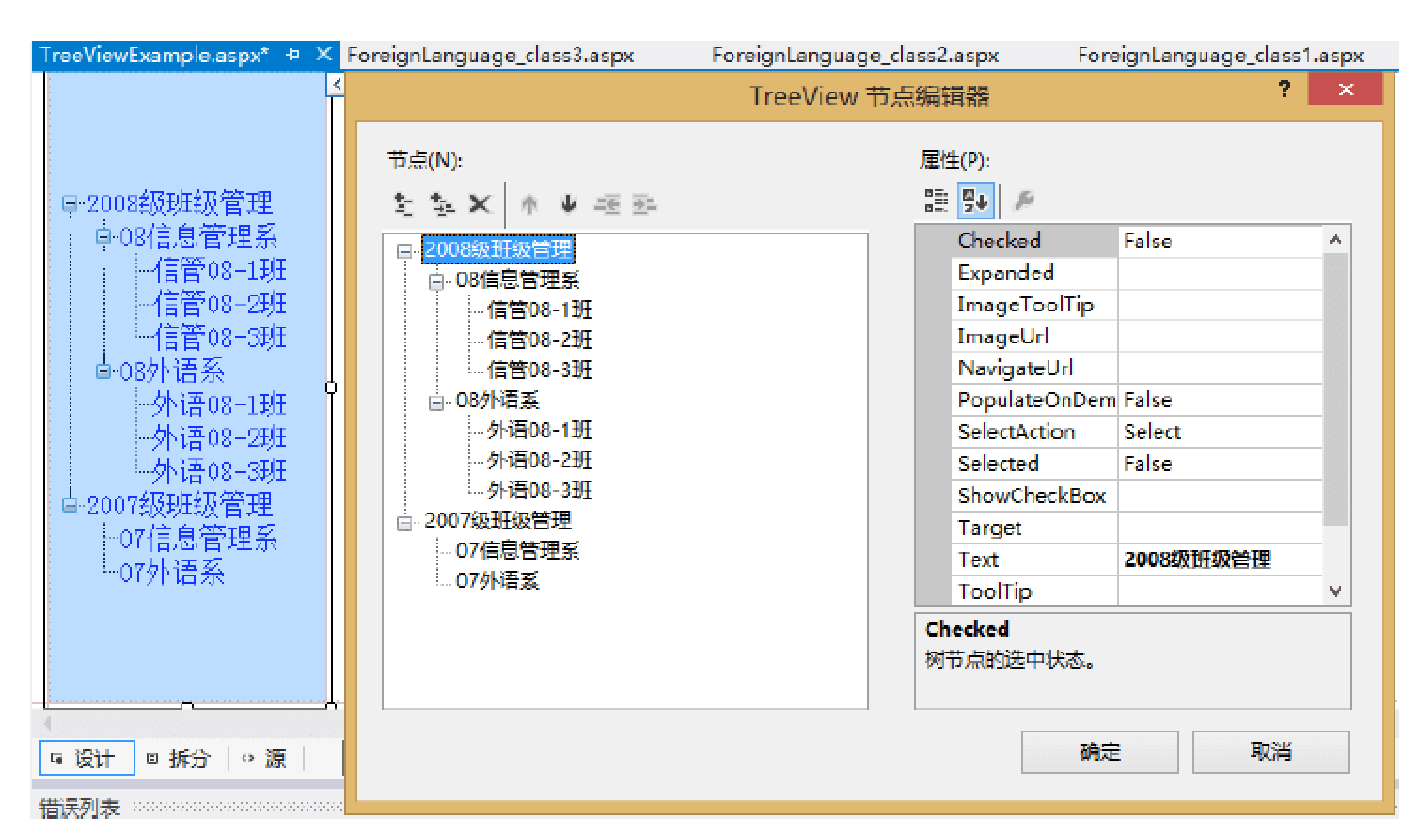


图 6-16　编辑 Treeview 节点

(6) 在【TreeView 节点编辑器】对话框右侧的属性选项中，利用 NavigateUrl 属性设置各节点链接的网页，全部设置完成后，单击【确定】按钮。

(7) 切换到 TreeViewExample.aspx 的【源】视图，将<body>和<body/>之间的部分改为如下内容。

```
    <asp:TreeView ID="TreeView1" runat="server"   Target ="_blank"   Height ="376px"
                Width ="165px" ShowLines="True">
                <Nodes>
                    <asp:TreeNode Text="2008 级班级管理" Value="2008 级班级管理">
                        <asp:TreeNode Text="08 信息管理系" Value="08 信息管理系">
                            <asp:TreeNode Text="信管 08-1 班" Value="信管 08-1 班"
NavigateUrl="~/InformationManage  class1.aspx" >
                            </asp:TreeNode>
                            <asp:TreeNode Text="信管 08-2 班" Value="信管 08-2 班"
NavigateUrl="~/InformationManage_class2.aspx" >
                            </asp:TreeNode>
                            <asp:TreeNode Text="信管 08-3 班" Value="信管 08-3 班"
NavigateUrl="~/InformationManage  class3.aspx">
    </asp:TreeNode>
                        </asp:TreeNode>
                        <asp:TreeNode Text="08 外语系" Value="08 外语系">
                            <asp:TreeNode Text="外语 08-1 班" Value="外语 08-1 班"
NavigateUrl="~/ForeignLanguage  class1.aspx">
    </asp:TreeNode>
                            <asp:TreeNode Text="外语 08-2 班" Value="外语 08-2 班"
```

```
NavigateUrl="~/ForeignLanguage_class2.aspx">
    </asp:TreeNode>
                            <asp:TreeNode Text="外语 08-3 班" Value="外语 08-3 班"
NavigateUrl="~/ForeignLanguage_class3.aspx">
    </asp:TreeNode>
                        </asp:TreeNode>
                    </asp:TreeNode>
                   <asp:TreeNode Text="2007 级班级管理" Value="2007 级班级管理">
                      <asp:TreeNode Text="07 信息管理系" Value="07 信息管理系">
    </asp:TreeNode>
                      <asp:TreeNode Text="07 外语系" Value="07 外语系">
    </asp:TreeNode>
                    </asp:TreeNode>
                </Nodes>
     </asp:TreeView>
```

(8) 为了便于区别此例中的各个网页，分别在 InformationManage_class1.aspx、InformationManage_class2.aspx、InformationManage_class3.aspx 网页中添加文字“欢迎来到信息管理系 08-1 班”、“欢迎来到信息管理系 08-2 班”和“欢迎来到信息管理系 08-3 班”。接着用同样的方法分别在 ForeignLanguage_class1.aspx、ForeignLanguage_class2.aspx、ForeignLanguage_class3.aspx 网页中添加文字“欢迎来到外语系系 08-1 班”、“欢迎来到外语系系 08-2 班”和“欢迎来到外语系系 08-3 班”。

(9) 切换到 TreeViewExample.aspx 网页，按 F5 键调试运行，分别展开和折叠相应节点，并单击“2008 级班级管理”下“08 信息管理系”和“08 外语系”中的叶节点，观察链接效果。

6.3　本章小结

一个网站通常是由很多网页组成。浏览者在一个网站中浏览页面时，是否能够容易地跳转到需要的页面，是否能够容易地获知当前浏览位置及上一个页面、下一个页面等十分重要。因此，本章主要对网站建设过程中常用的页面切换方法、页面之间传递数据的方法，以及网站导航功能，通过站点地图、SiteMapPath、Menu、TreeView 等实现网页导航的方法进行了详细介绍。通过本章的学习，开发人员能够对如何设计一个良好易用的网站应用程序有个更深刻的认识。

6.4　思考和练习

1. 页面切换的方式有几种？分别是什么？
2. 页面间数据传递的方式有几种？分别是什么？
3. 网站导航的方式有几种？分别是什么？

第7章　样式、主题与母版页

开发 Web 应用程序通常需要考虑两个方面：功能和外观。其中，外观考虑的重点是使 Web 站点的美观问题，如控件的颜色、图像的使用、页面的布局等。ASP.NET 提供了一些自定义功能，主要用在 ASP.NET 程序中的页面、控件的外观及样式上，例如，可以为某个控件设置字体、背景色和前景色、宽度以及高度等样式，还可以通过合理定位页面中的元素，为用户呈现出易于使用且视觉美观的效果。本章将全面介绍 Web 应用程序的样式控制和页面布局。

本章的学习目标：

- 理解 CSS 的概念，掌握 CSS 的用法。
- 理解布局的概念，掌握 CSS 和 Div 布局的方法。
- 理解主题的概念，掌握主题的创建和引用。
- 理解母版页和内容页的概念，掌握创建母版页和内容页的方法。

7.1　CSS 简介

样式是指 XHTML 标记元素在浏览器中呈现的形式，如字体的大小、颜色、位置等。在 XHTML 中，最容易、最快速的方法是通过设置标记元素的 style 属性来控制元素的样式，其一般形式如下。

```
<元素名称  style="属性名 1:属性值 1;  属性名 2:属性值 2;  ……" >显示内容</元素名称>
```

属性名与属性值之间用冒号(:)分隔，如果一个样式中包含多个属性，各属性之间用分号(;)隔开。示例代码如下。

```
<div style="font-weight:bold;color:Red;border:solid 2pt navy">
```

这是一个有样式的层</div>，在 Internet Explorer 中显示效果如图 7-1 所示。

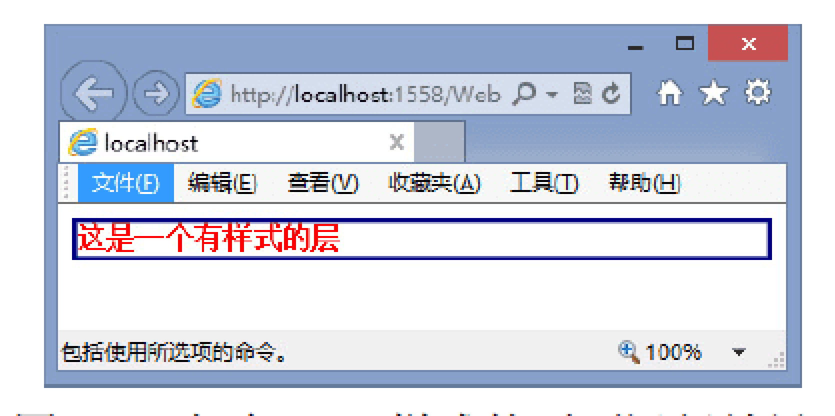


图 7-1　包含 style 样式的页面运行效果

这种直接通过设置标记元素的 style 属性来控制样式的方式称为内联样式，它的优点就是直观、方便。但是，这种方式的缺点也很明显，如果有多个元素具有相同的样式，那么必须进行多次重复设置，而且修改样式非常繁琐，需要逐一对每个元素进行修改。因此，内联样式不适合控制具有大量重复样式的元素，所以需要使用其他更简便的方法来提高效率。

本节就来介绍 CSS 的概念，以及如何利用 VS 定义 CSS，CSS 的样式规则和用法。

7.1.1　CSS 语言

CSS(Cascading Style Sheet)，即层叠样式表，是一种标记性语言，用于控制网页样式并允许将样式信息与网页内容分离。使用 CSS 样式可以非常灵活地控制网页外观，大大减轻实现精确布局定位、维护特定字体和样式的工作量。

CSS 中有两种定义样式的方法，分别是内联式和级联式，分别对应两类样式：内联式样式和级联式样式。

1. 内联式样式

直接将样式控制放在单个 HTML 元素内，称为内联式或行内样式。该样式通过 style 属性控制每个元素的外观。这种样式直观但是很繁琐，除非具有相同样式的元素较少，否则很少采用。下面的代码采用内联式来控制各个元素的样式。

【例 7-1】使用 style 样式化一个简单页面。

启动 VS，新建 WebSite7 网站，然后新建 example7-1 页面，修改<body></body>标签对的代码如下。

```
<body style="text-align:center">
<form id="form1" runat="server">
<div style="text-align:center; width:400px; border:solid 1px blue">
<h1 style="font-size:x-large; color:red ">欢迎光临</h1>
<h2 style="font-size:large; color:blue ">这是一个被 style 修饰的页面</h2>
</div>
</form>
</body>
```

2. 级联式样式

在网页的 head 部分定义或导入的样式，称为级联式样式。该样式可以实现将网页业务处理逻辑和呈现代码分离，这样，当修改某些元素的样式时，只需要修改 head 部分定义或引入的样式，该网页内所有具有相同样式的元素都会自动应用定义的或引入的样式。

级联式样式可以使用以下两种方式来实现。

(1) 第一种方式是，在 head 部分直接定义 CSS 样式，称为内嵌式。即这种 CSS 一般位于 HTML 文件的头部，即<head>与</head>标签内，并且以<style>开始，以</style>结束。例如，将【例 7-1】代码中的样式抽取出来，经过修改得到如下代码。

```
<head>
 <title>内嵌式样式</title>
<style type="text/css">
<!--
body{ text-align:center }
div{ text-align:center; width:400px; border:solid 1px blue }
h1{ font-size:x-large；color:red}
 h2{ font-size:large;   color:blue }
```

```
-->
</style>
</head>
<body>
<form id="form1" runat="server">
<div>
<h1>欢迎光临</h1>
<h2>这是一个被 style 修饰的页面</h2>
</div>
</form>
</body>
```

其中<style>与</style>直接是样式的内容，在{ }前面可以写样式的类型和名称，{ }中是样式的属性。这种方法是最常用的添加样式表的方法。

可见，采用内嵌式比内联式方便了很多，body 内的代码也相对简洁，修改某个元素的样式时只需要修改 head 内的代码即可。但是，内嵌式只解决了一个网页内部结构和表现分离的问题，一般情况下网站是由很多网页组成，如果不同网页中的某些元素采用了相同的样式，仍然需要分别设置，因此，将样式放在一个单独的 CSS 文件中，然后通过为每个网页引入该文件来实现统一的外观成为一种更好的选择。这就是接下来要介绍的另一种使用样式表的方式。

(2) 第二种方式是，在 head 部分通过导入以扩展名为.css 的文件来实现 CSS 样式，称为链接式。利用这种方法在网页中可以调用已经定义好的样式表来实现样式表的应用，定义好的样式表通常单独以文件的形式存放在站点目录中。这种方法实现了将网页结构和表现的彻底分离，最适合大型网站的 CSS 样式定义。

例如，将【例 7-1】代码中的样式抽取出来保存为 Style.css，文件内容如下。

```
body
{
    text-align:center;
}
div
{
    text-align:center;
    width:400px;
    border:solid 1px blue;
}
h1
{
    font-size:x-large;
    color:red;
}
h2
{
    font-size:large;
    color:blue;
}
```

引用代码如下：

```
<head>
    <title>链接式样式</title>
    <link href="Style.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <h1>欢迎光临</h1>
            <h2>这是一个被 style 修饰的页面</h2>
        </div>
    </form>
</body>
```

在引用样式的标记<link>中，ref 属性规定了 XHTML 与被链接文件的关系，href 属性指定了要链接的样式表文件的 URL，type 属性则规定了链接文件的类型。

此外，如果某个元素既引用了链接样式文件中定义的样式，又在 head 部分定义了新的样式，或者在元素内部通过 style 属性定义了新的样式，那么该标记元素最终呈现的效果会是什么样呢？下面通过一个例子来说明。

【例 7-2】样式嵌套举例。

启动 VS，打开 WebSite7，新建名为 example7-2.aspx 的页面，其中关键代码如下：

```
<head>
<title>链接式样式</title>
<style type="text/css">
<!--
h1{ font-weight:bold }
   h2{ color: yellow}
-->
</style>
<link rel="stylesheet" href="style.css" type="text/css" />
</head>
<body>
<form id="form1" runat="server">
<div>
<h1 style=" font-size:small ">欢迎光临</h1>
<h2 style=" font-weight:bold ">这是一个被 style 修饰的页面</h2>
</div>
</form>
</body>
```

其中，style.css 文件的内容同上。运行这个 XHMTL 文件，在浏览器中可以看到，h1 元素内的文字以粗体、小号、红色显示，而 h2 元素内的文字则以粗体、大号、蓝色显示。

可见，链接式样式中 h1 元素的 font-size 属性和内嵌式样式中 h2 元素的 color 属性都没有起作用，而不冲突的样式则都会起作用。这就是样式嵌套中的冲突问题，浏览器解决这种问题的方法就是一旦发现样式冲突，则通过“就近使用”原则，采用距离该元素最近

的样式进行显示，而不冲突的样式则通过顺序组合后形成最终样式进行显示。

设计者可以根据实际情况选择一种或多种样式控制方法进行样式定义。一般情况下，在样式表(.css)文件中定义适合大多数网页公用的样式，在网页内部采用内嵌式定义该页面特有的样式，内联式样式定义个别元素的样式，再结合可视化的开发工具，从而使样式控制真正灵活、方便。

在 VS 中，设置样式的方法有两种，一种是在【源】视图下直接设置样式，另一种是利用可视化界面设置样式。

7.1.2　代码方式引用样式

创建一个新的网页，系统自动打开页面【源】视图，利用系统提供的智能提示功能，可以方便地设置各种元素的样式。下面通过具体步骤说明样式设置的方法。

(1) 在 example7-1.aspx 源文件的 body 部分输入【例 7-1】的代码，输入时可以看到 VS 所提供的智能提示工具，如图 7-2 所示。

(2) 完成代码输入后切换到【设计】视图，可以查看页面的显示效果，如图 7-3 所示。

虽然在【源】视图下输入样式信息时，系统将提供智能提示以帮助用户完成样式输入，但是初学者往往不了解到底都有哪些样式可供使用，对于提供的选项也不知道什么含义，因此，利用 VS 的可视化界面可以更为方便地完成样式的定义。

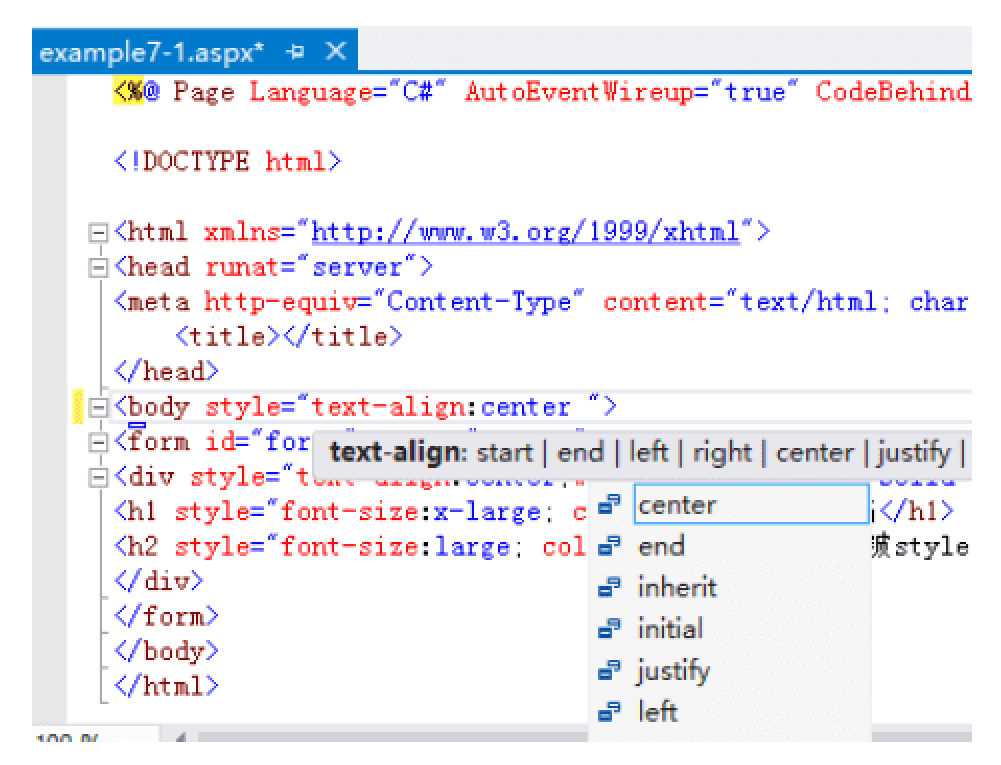


图 7-2　源视图中提供的智能代码提示

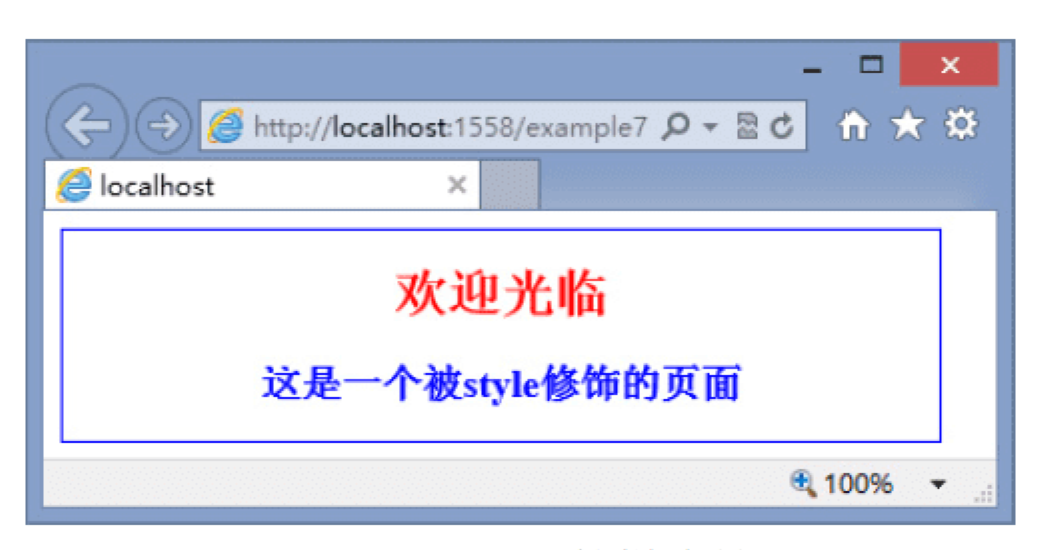


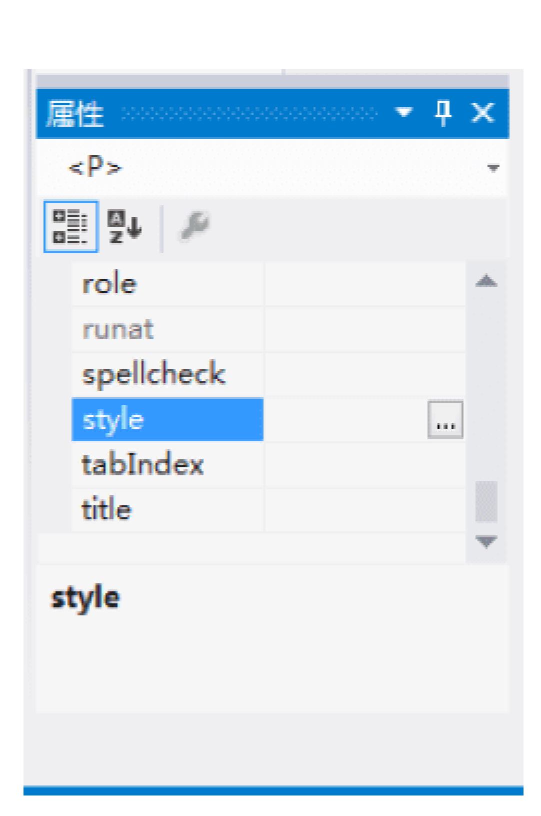
图 7-3　设计视图下的样式效果

7.1.3　可视化方式引用样式

利用可视化窗口设置样式的方法有很多，可以在【源】视图或者【设计】视图下选中某个标记元素，然后在属性面板中 style 属性值的某个地方单击，将在样式信息后看到省略号按钮(…)，如图 7-4 所示。单击该按钮将看到【修改样式】对话框，如图 7-5 所示。

该对话框分为两个窗格，左窗格列出 9 个类别，当选择某个类别时，右窗格显示所选类别下的选项。选择了样式选项并单击【确定】按钮后，新的样式定义将自动在【源】视图中生成，还可以在【设计】视图下查看最新的效果。

这种方法只能将定义的样式属性以内联式生成，放在每个元素的 style 属性中，内联式样式的缺点前面已经讲过。要想定义内嵌式样式也非常容易，具体步骤如下。





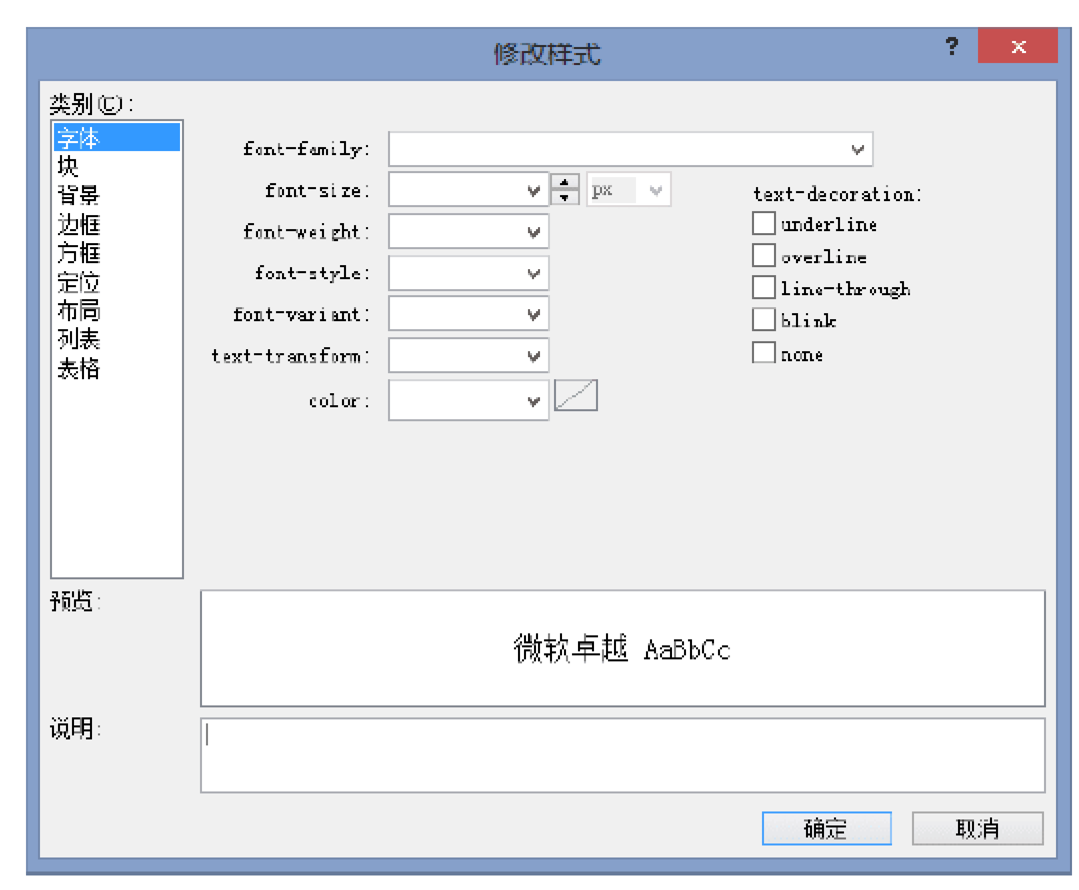


图 7-4　属性面板的 style 属性　　　　图 7-5　【修改样式】对话框

(1) 切换到【设计】视图中，在【样式应用】工具栏的【目标规则】列表中，单击【应用新样式】，如图 7-6 所示。

(2) 此时将显示【新建样式】对话框，该对话框和如图 7-5 所示的【修改样式】对话框相似，不同的是【新建样式】对话框中包含了选择器用于选择对哪一个标记进行定义，以及通过定义位置将当前定义存放到指定位置，如图 7-7 所示。

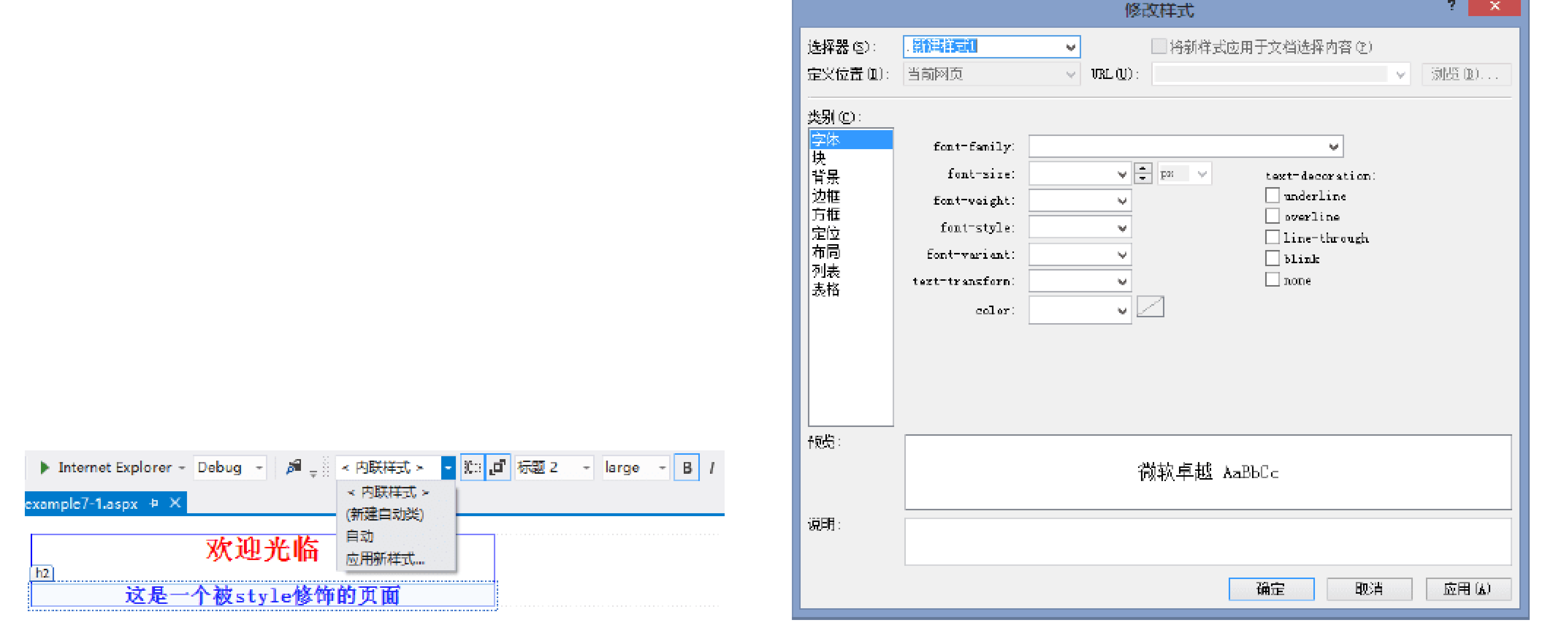


图 7-6　新建样式　　　　图 7-7　【新建样式】对话框

在【选择器】列表中单击 h1，就可以创建应用于所有 h1 元素的样式。

注意，【定义位置】列表设置为“当前网页”，这指示该样式规则在当前页的 style 元素中创建。若想查看已创建的样式规则，可以切换到【源】视图并滚动到 style 元素，该元素位于 head 元素内。

也可以通过选择菜单【格式】中的【新建样式】命令来定义内嵌式样式。

想要对正在定义的内嵌式样式规则进行修改也非常简单，具体步骤如下。

(1) 单击某个想要修改样式的元素，如文本“欢迎光临”。此时，选中部分以蓝色框包围并有一个标签指示 h1 元素已选中。

(2) 选择【视图】菜单中的【CSS 属性】命令，打开【CSS 属性】面板，可以看到 h1 元素的 CSS 属性列表，如图 7-8 所示。

在 CSS 属性列表中所做的修改可以立即通过【设计】视图显示出来，如果想要观察具体的样式代码，可以通过切换到【源】视图并滚动到 style 元素来进行查看。

使用 CSS 的另一个有效方法是将样式规则放入样式表中。然后，所有页面都可以引用这些样式，这样可以使这些页面看起来非常一致。创建样式表使用的技术与创建新页面所用的相同。具体步骤如下。

(1) 在【解决方案资源管理器】中，右击网站的名称，然后单击【添加新项】。在【Visual Studio 已安装的模板】下选择【样式表】。

(2) 在【名称】文本框中，输入 NewStyle.css，然后单击【添加】按钮。编辑器将打开，其中显示一个包含空 body 样式规则的新样式表。

(3) 打开或切换到 example7-2.aspx 页，然后切换到【设计】视图。在【视图】菜单中单击【管理样式】命令，打开【管理样式】面板。单击【附加样式表】，此时将显示【选择样式表】对话框，选择 NewStyle.css 文件，然后单击【确定】按钮，名为 NewStyle.css 的新选项卡在【管理样式】窗口中创建，如图 7-9 所示。

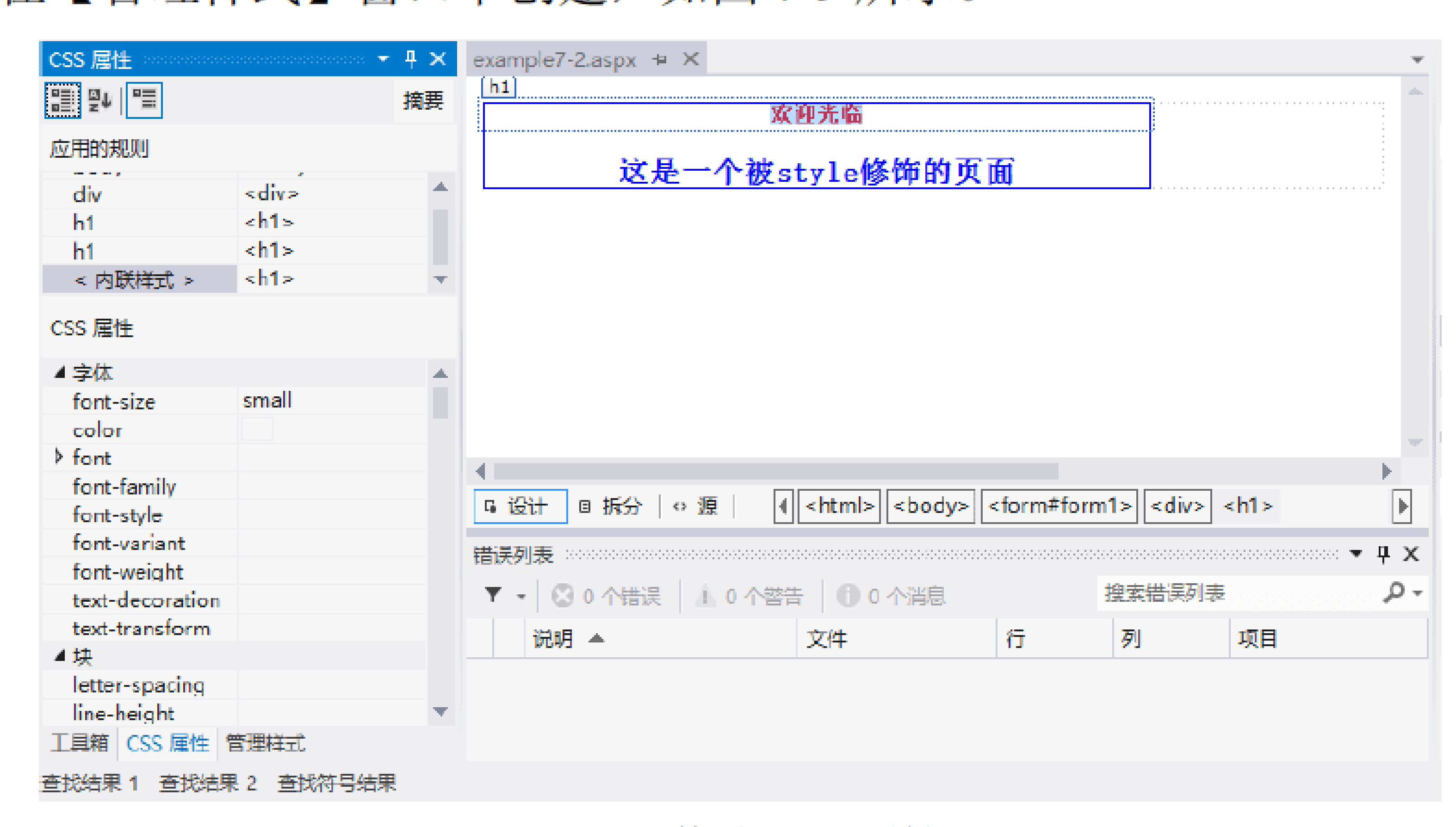


图 7-8　修改 CSS 属性



图 7-9　新建 CSS 样式表文件

可以通过多种方式为页面指定样式表。最简单的方法是在【源】视图中将文件从【解决方案资源管理器】拖到页面的头元素中或者直接将文件拖到【设计】视图中。

7.1.4　样式规则

从生成的样式代码可以看到，无论是定义内嵌式样式还是链接式样式，每个样式的定义格式都是一样的，如下所示。

```
样式定义选择符{ 属性 1:值 1; 属性 2:值 2; ……}
```

其中，样式定义选择符是指样式定义的对象，可以是 HTML 标记元素，还可以是用户自定义的类、用户自定义的 ID、伪类和伪元素等。

1. 标记选择符

任何 HTML 元素都可以是一个 CSS 的标记选择符。标记选择符仅仅是指向特别样式的元素。例如下面规则中的标记选择符是 div。

```
div{  text-align:center }
```

该样式规则的含义是，div 块内的所有内容都居中显示。图 7-10 中的【选择器】下拉列表中提供了所有可供使用的标记选择符。

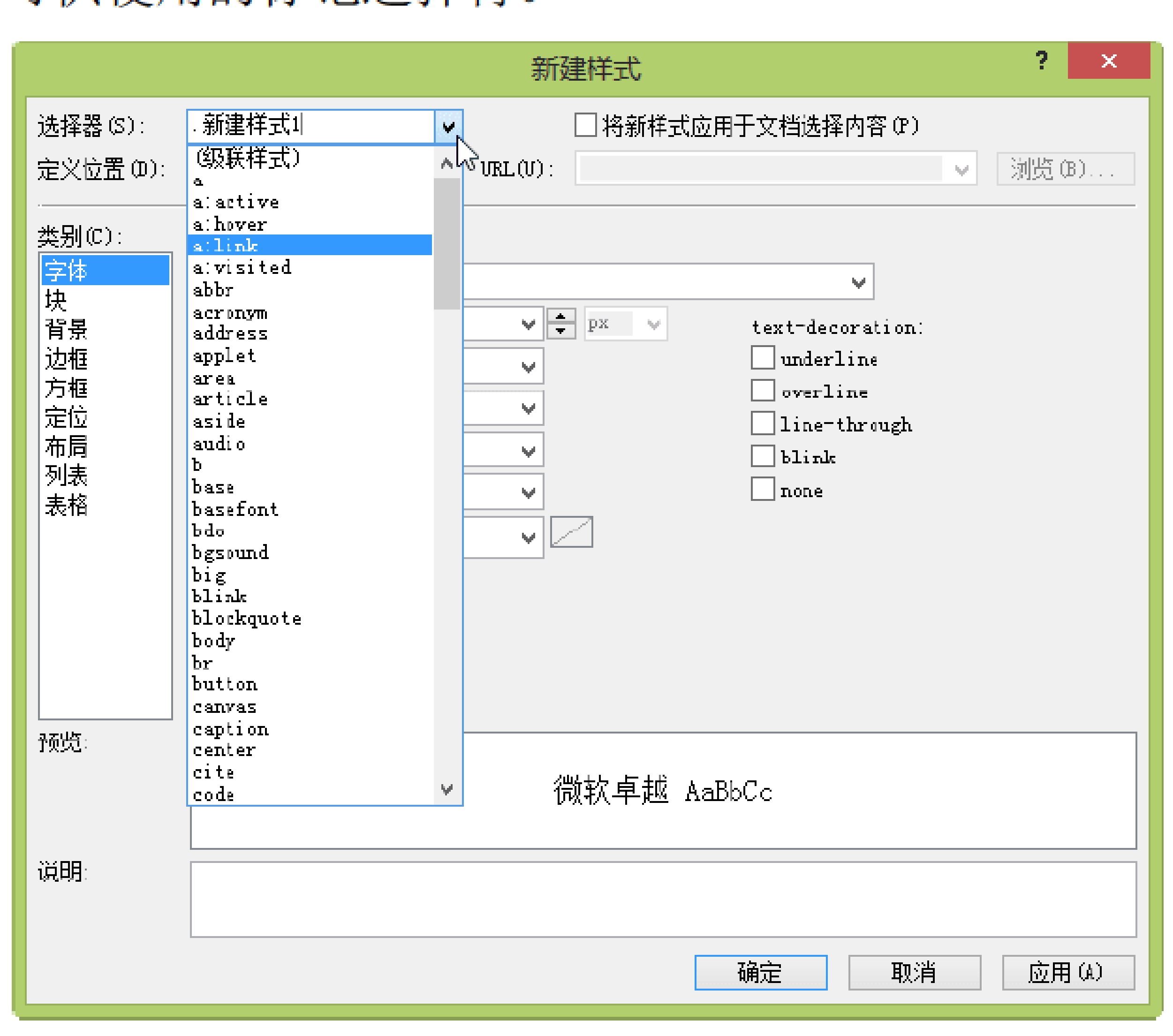


图 7-10　【选择器】下拉列表

2. 类选择符

每一个标记选择符都能自定义不同的类，从而允许同一元素具有不同的样式。指定某个标记选择符内的自定义类的一般形式如下。

```
标记选择符.类名{样式属性 1:值 1; 样式属性 2:值 2; ……}
```

示例代码如下。

```
<style type="text/css">
p.one{
        color:red;
```

```
        }
    p.two{
            color:blue;
        }
    </style>
```

在代码中引用类选择符的方法是通过元素的 class 属性来实现的，示例代码如下。

```
<p class="one">类别选择器 1</p>
<p class="two">类别选择器 2</p>
```

其含义是在 p 中引用 one 会以红色样式显示，在 p 中引用 two 会以蓝色样式显示。

类选择符的定义也可以与标记选择符无关，这样，类选择符可以应用于任何元素。这种自定义类选择符的形式如下。

```
.类名{样式属性 1:值 1; 样式属性 2:值 2; ……}
```

示例代码如下。

```
<style type="text/css">
.note{
color:red;
}
</style>
<h1 class="note">类别选择器 1</p>
<h2 class="note">类别选择器 2</p>
```

在这个例子，note 类选择符可以被用于任何元素。

3. ID 选择符

ID 选择符用于分别定义每个具体元素的样式。一个 ID 选择符的指定要有指示符#在名字前面。使用时通过指定元素的 id 属性来关联。示例代码如下。

```
#index { color:blue }
```

引用时，使用 id 属性声明即可。

```
<p id="index">本段落的颜色为灰色</p>
```

可以看出，自定义 ID 选择符与自定义类选择符的定义方式非常相似，但是两者在使用上是有区别的。在同一个网页中，多个标记元素可以使用同一个自定义类选择符，而 ID 选择符只能为某一个标记元素使用。这种选择符应该尽量少用，因为它具有一定的局限。但是如果在一个元素的样式定义中，既有标记选择符，又有自定义类选择符和自定义 ID 选择符，那么自定义 ID 选择符的优先级最高，其次是自定义类，标记选择符的优先级最低。

4. 伪类

伪类是 CSS 中非常特殊的类，能自动地被支持 CSS 的浏览器所识别。伪类可以指定 XHTML 中的 A 元素以不同的方式显示链接(links)、已访问链接(visited links)和可激活链接

(active links)。其中，一个已访问链接可以定义为不同颜色的显示，甚至不同字体大小和风格。

CSS 中用 4 个伪类来定义链接的样式，分别是：a:link、a:visited、a:hover 和 a : active，例如：

```
a:link{font-weight : bold ;text-decoration : none ;color : #C00000 ;}
a:visited {font-weight : bold ;text-decoration : none ;color : #C30000 ;}
a:hover {font-weight : bold ;text-decoration : underline ;color : #F60000 ;}
a:active {font-weight : bold ;text-decoration : none ;color : #F90000 ;}
```

以上语句分别定义了“链接、已访问过的链接、鼠标停在上方时、点下鼠标时”的样式。注意，必须按以上顺序写，否则显示可能和预想的不一样。记住它们的顺序是 LVHA。

5. 关联选择符

关联选择符是一个用空格隔开的两个或更多的单一标记选择符组成的字符串。一般格式如下。

```
选择符 1　选择符 2　…… {属性:值; ……}
```

这些选择符具有层次关系，并且它们的优先级比单一的标记选择符大。如下所示。

```
p h1{ color:red }
```

这种定义方式只对 p 所包含的 h1 元素起作用，单独的 p 或者单独 h1 元素均无法采用该样式。

6. 并列选择符

如果有多个不同的元素定义的样式相同，则可以使用并列选择符简化定义。如下所示。

```
h1,h2,h3{ color:blue}
```

每个元素之间用逗号隔开，表示所有的 h1、h2、h3 标记中的内容都将以蓝色显示。

7.2　页面布局

除了为页面的内容设置样式，页面元素的布局和定位是否合理也是衡量网页设计是否美观的重要指标。本节将介绍网页的基本布局方式，页面元素的定位，以及表格布局和层布局。

7.2.1　网页的基本布局方式

常见的网页布局方式有左对齐、居中和满宽度显示。默认情况下，网页内容水平左对齐，然而，在实际页面中，经常看到的大部分页面都是水平居中和满宽度显示的。因此，这里仅介绍两种非常常用的布局实现方法。

1. 页面水平居中

实现的方法是在 body 的 style 样式中设置 text-align 属性的值为 center 即可。如果还希望页面的宽度固定，则可以通过设置 div 的 width 属性来实现。示例代码如下。

```
<body style="text-align:center; ">
  <form id="form1" runat="server">
  <div id="div1" style="width:760px; text-align:center; height:500px"></div>
  </form>
</body>
```

2. 页面满宽度显示

实现的方法是将 div1 的固定宽度设置为百分比即可，这样宽度就会随显示界面的大小自动调整。示例代码如下。

```
<body style="text-align:center; ">
  <form id="form1" runat="server">
 <div id="div1" style="width:98%; text-align:center; height:500px"></div>
</form>
</body>
```

这种方式的优点是无论浏览器是否最大化显示，都不会出现横向滚动条；缺点是页面元素相对位置不固定，不利于用户和窗体之间的操作。

7.2.2　页面元素定位

页面元素的定位分为流布局和坐标定位布局两种，其中，坐标定位布局又分为绝对定位和相对定位，这里仅介绍流布局和坐标绝对定位。

1. 流布局 static

如果采用该布局，则页面中的元素将按照从左到右、从上到下的顺序显示，各元素之间不能重叠。如果不设置元素的定位方式，则默认是流式布局。

2. 坐标绝对定位 absolute

在使用坐标绝对定位之前，必须先将 style 元素的 position 属性的值设置为 absolute，然后就可以由 style 元素的 left、top、right、bottom 和 z-index 属性来决定元素在页面中的绝对显示位置。left 属性表示元素的 x 坐标，top 属性表示元素的 y 坐标，坐标的位置是以它最近的具有 position 属性的父容器为参照物的。具体效果参看下面的例子。

【例 7-3】在 WebSite7 网站中新建一个 example7-3.aspx 页面，修改页面<body>内的代码为下面的内容。

```
<body>
  <form id="form1" runat="server">
    <div id="div1" style="border: 1px #000080 solid; text-align: left; width: 400px; height: 200px;">
      <div id="div2" style="width: 200px; height: 120px; text-align: left; border: 1px #00FF00 solid;
```

```
background-color: #808080">
            <div id="div3" style="position: absolute; top: 70px; left: 130px; width: 150px; height: 100px;
border: 2px #800000 solid; background-color: #FFFF00">
              <div id="div4" style="position: absolute; left: 30px; top: 30px; width: 100px; height: 60px;
border: 3px #FF00FF solid; background-color: #00FFFF">
              </div>
            </div>
          </div>
        </div>
      </form>
    </body>
```

然后切换到【设计】视图，观察显示的效果，如图 7-11 所示。运行该页面，可以看到，无论浏览器窗口如何变化，各层之间的位置仍然保持不变。

具有不相同 z-index 值的元素可以重叠，其效果就像多张透明的纸按顺序叠放在一起。其中，z-index 值大的元素会覆盖 z-index 值小的元素。修改【例 7-3】的代码，为 div3 增加 z-index 属性，如下所示。

```
    <div id="div3" style="position: absolute;    ……    z-index: -1;">
```

切换到【设计】视图，观察显示的效果，如图 7-12 所示。

采用坐标定位的方式可以精确地将元素放在页面中相应的位置显示，但是由于不同浏览器在显示方面存在的差异，也会给整体页面布局带来混乱的效果，解决这个问题的方法就是利用表格来进行布局。

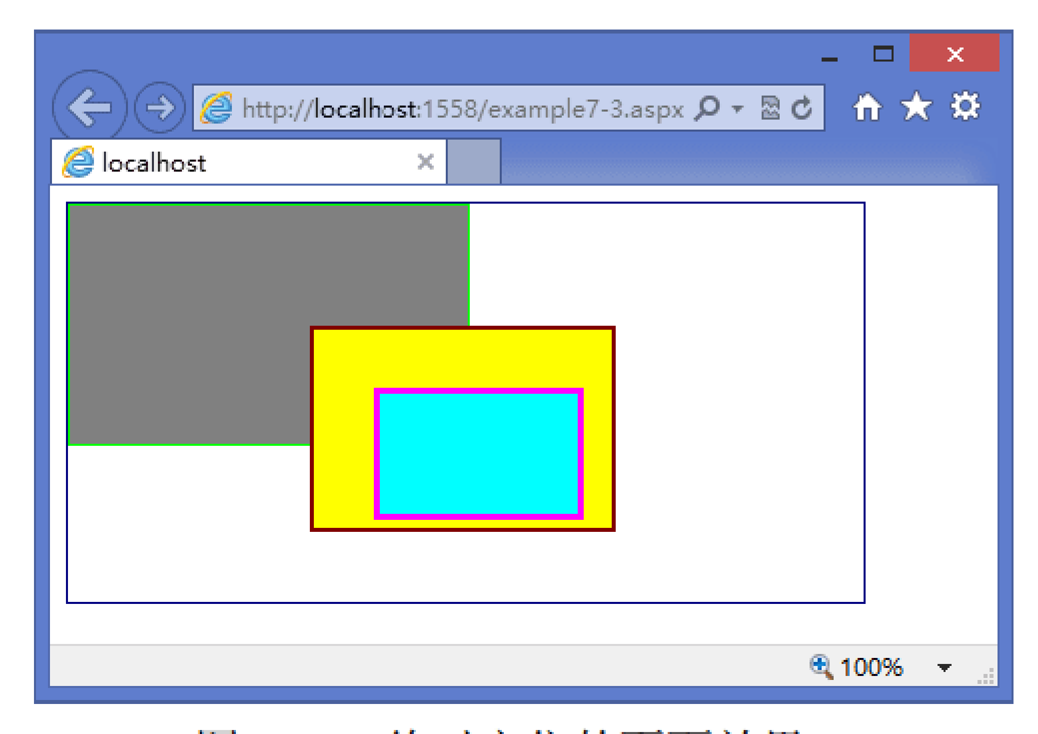


图 7-11　绝对定位的页面效果

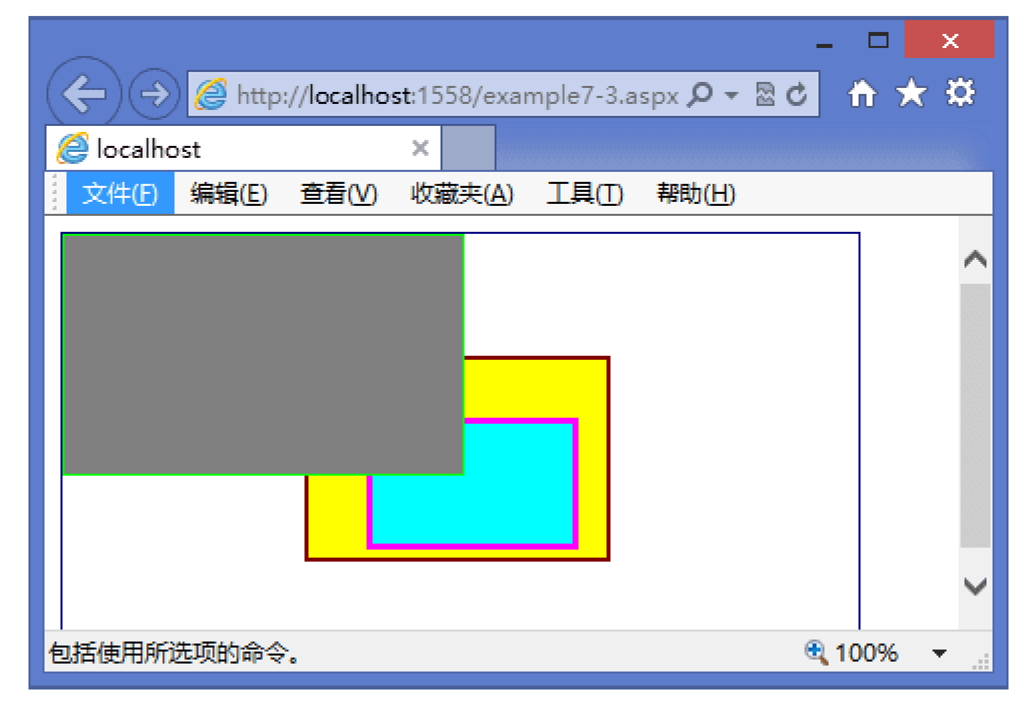


图 7-12　设置 z-index 属性的效果

7.2.3　表格布局

利用表格可以将网页中的内容合理地放置在相应的区域，每个区域之间互不干扰。例如，设计一个表格用来布局网页首页，实现的效果如图 7-13 所示。

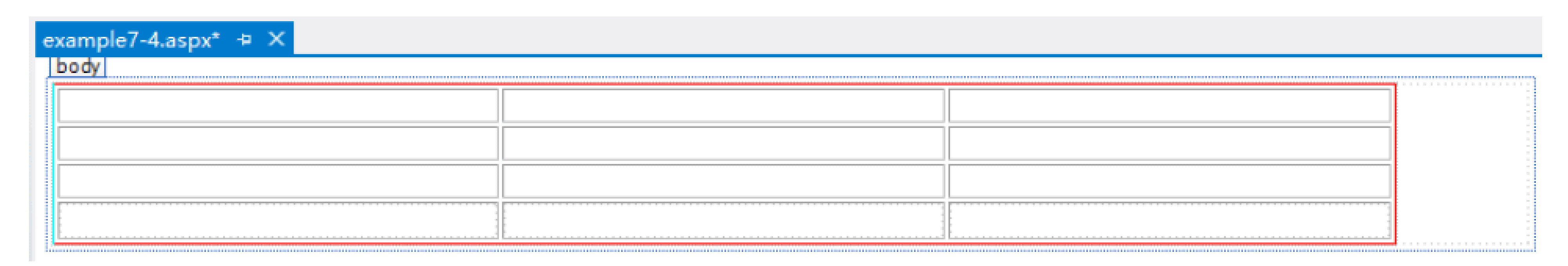


图 7-13　表格布局

从图 7-16 中可以看到，表格中定义了 1 个标题区，1 个导航区，1 个页脚区，中间又分成 3 个区，这就需要先创建一个 4 行 3 列的表格，然后再通过详细设置达到图 7-13 中的效果。该实例的实现步骤如下。

(1) 在【解决方案资源管理器】中，右击网站的名称，然后单击【添加新项】，新建一个 Default.aspx 页面，设置 body 元素的 style 属性为"text-align:center"，div 元素的 style 属性为"width: 780px; text-align:center"。

(2) 切换到【设计】视图，将光标停在 div 标记内。选择菜单【表】中的【插入表】命令，打开【插入表格】对话框，定义表格大小为 4 行 3 列，指定宽度为 100%，边框值为 1，边框颜色为红色。如图 7-14 所示。

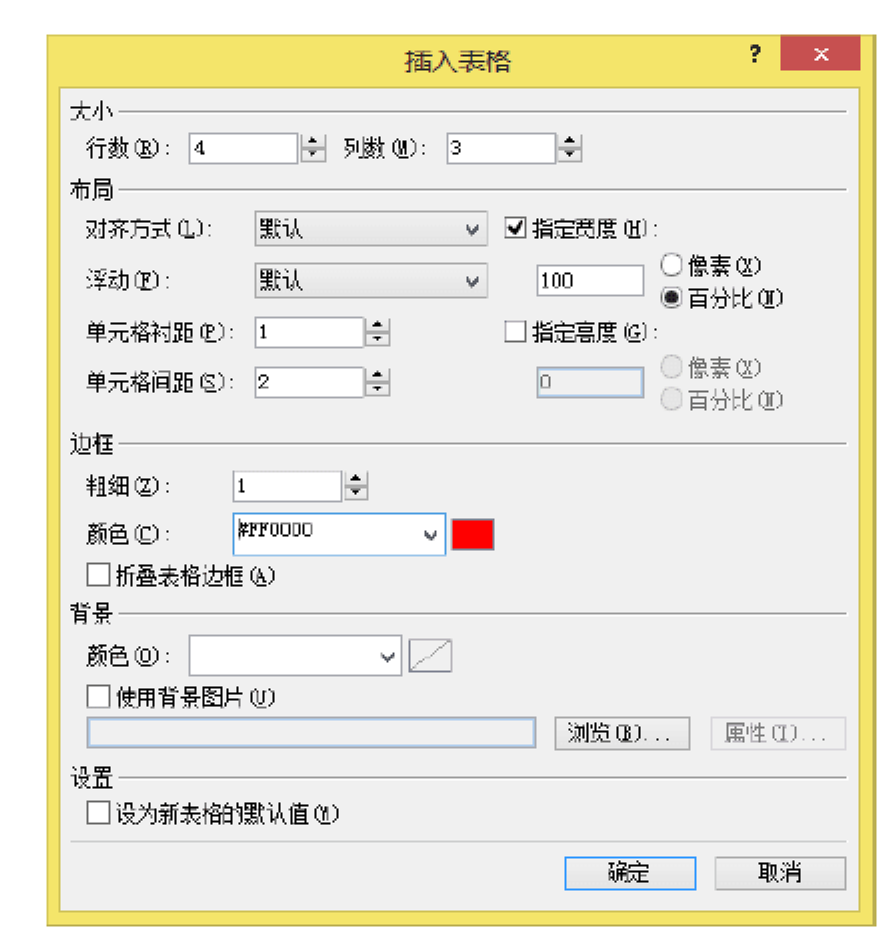


图 7-14　新建表格

经过详细设置后的代码可以通过切换到【源】视图中查看，如【例 7-4】所示。

【例 7-4】表格布局举例。

```
<head runat="server">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title></title>
    <style type="text/css">
        .auto-style4 {
            width: 100%;
            border: 1px solid #FF0000;
        }
        .auto-style5 {
            height: 20px;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
    <div style="width: 780px; text-align:center">

        <table class="auto-style4" border="1">
            <tr>
                <td> </td>
                <td> </td>
                <td> </td>
            </tr>
            <tr>
                <td> </td>
                <td> </td>
                <td> </td>
            </tr>
            <tr>
```

```
                <td> </td>
                <td> </td>
                <td> </td>
            </tr>
            <tr>
                <td class="auto-style5"></td>
                <td class="auto-style5"></td>
                <td class="auto-style5"></td>
            </tr>
        </table>
    </div>
    </form>
</body>
```

整个表格的最大列数是 3，所以标题区、导航区和页脚部分的 colspan 属性值为 3，表示对应行占 3 列。此时，浏览器会自动根据整个表格的宽度调整该部分的宽度。

表 7-1 中列出了表格中部分常用的属性。

表 7-1　常用表格属性

属 性 名	含　义
Border	表示边框宽度，如果设置为 0，表示无边框，此时默认 frame=void，rules=none；可以设置为大于 0 的值来显示边框，此时默认 frame=border，rules=all
Cellspacing	表示单元格间距(表格和 tr 之间的间隔)
Cellpadding	表示单元格衬距(td 和单元格内容之间的间隔)
Frames	表示如何显示表格边框，void：无边框(默认)；above：仅有顶部边框；below：仅有底部边框；hsides：仅有顶部和底部边框；vsides：仅有左右边框；lhs：仅有左边框；rhs：仅有右边框；box 和 border：包含全部 4 个边框
Rules	表示如何显示表格内的分隔线，all：显示所有分隔线；cols：仅显示列线；rows：仅显示行线；groups：仅显示组与组之间的分隔线

表格布局的最大优点就是简单直观。但是如果将整个网页的元素都包含在表格内，则浏览器会将整个表格全部下载完毕后才显示表格中的内容，因此网页显示速度慢。此外，表格布局也不利于网页结构和表现的分离。解决该问题的方法就是网页整体采用 DIV 和 CSS 进行层布局，局部用表格进行布局，这是当前 Web 标准推荐的最佳布局方法。

7.2.4　DIV 和 CSS 布局

层布局最核心的标签就是 DIV。DIV 是一个容器，在使用时以<DIV></DIV>形式存在。在 XHTML 中，每一个标签都可以称作是容器，能够放置内容。但 DIV 是 XHTML 中专门用于布局设计的容器对象。

在传统的表格布局中，完全依赖于表格对象 TABLE，在页面中绘制多个单元格，在表格中放置内容，通过表格的间距或者用无色透明的 GIF 图片来控制布局版块的间距，达到排版目的。而以 DIV 对象为核心的页面布局中，通过层来定位，通过 CSS 定义外观，最大程度地实现了结构和外观彻底分离的布局效果，因此习惯上对层布局又称为 DIV 和

CSS 布局。

1. 定义层

添加层的方法非常简单，可以通过【工具箱】面板中的 HTML 选项卡拖动一个 Div 项到设计视图中，或者在【源】视图中创建一对<div></div>标记。

【例 7-5】分析一个简单的定义 DIV 的例子。设计一个页面，添加一个层，定义其样式效果如图 7-15 所示。



图 7-15　简单的层定义

具体实现步骤如下。

(1) 在【解决方案资源管理器】中，右击网站 WebSite7 的名称，然后单击【添加】|【添加项】，新建一个 example7-5.aspx 页面，此时会发现代码中已经包含了一个层对象。

(2) 切换到【设计】视图，在【格式】菜单中单击【新建样式】命令，打开【新建样式】对话框，在【选择器】后面的文本框中输入#sample，然后选择相应的类别进行设置，完成后单击【确定】按钮。

(3) 选中层对象，在【视图】菜单中单击【管理样式】命令，然后右击#sample 样式，选择【应用样式】命令即可，如图 7-16 所示。

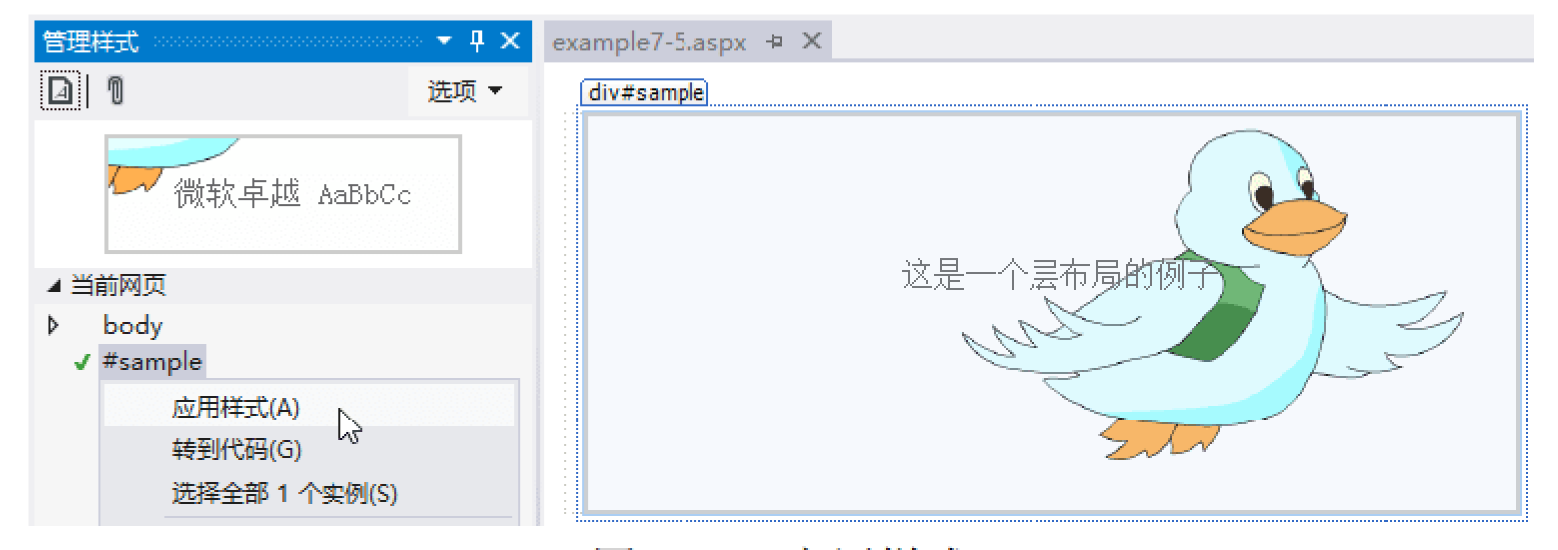


图 7-16　应用样式

对应的程序代码如下所示。

```
<head id="Head1" runat="server">
   <title></title>
   <style>
      body{ text-align:center; }
      #sample
      {
          border: 2px solid #CCC;
              margin: 10px;
              padding: 20px 10px 10px 20px;
```

```
                background: #fefefe url('images/bg_image1.gif') no-repeat right bottom;
                color: #666;
                text-align: center;
                line-height: 120px;
            width:51%;
                height: 169px;
            }
    </style>
</head>
<body>
<form id="form1" runat="server">
<div id="sample">这是一个层布局的例子</div>
</form>
</body>
```

其中，margin 是指层的边框以外留的空白，用于页边距或者与其他层制造一个间距。10px 10px 10px 10px 分别代表“上右下左”(顺时针方向)4 个边距。如果都一样，可以缩写成 margin: 10px;。如果边距为零，则写成 margin: 0px;。

padding 是指层的边框到层的内容之间的空白。和 margin 一样，分别指定上右下左边框到内容的距离。如果都一样，可以缩写成 padding:0px。单独指定左边可以写成 padding-left: 0px;。padding 是透明元素，不能定义颜色。

border 是指层的边框，border-right: #CCC 2px solid;是定义层的右边框颜色为#CCC，宽度为 2px，样式为 solid 的直线。如果要虚线样式可以用 dotted。

background 是定义层的背景。分两级定义，先定义图片背景，采用 url(../images/bg_image1.gif)来指定背景图片路径；其次定义背景色#fefefe。no-repeat 指背景图片不需要重复，如果需要横向重复用 repeat-x，纵向重复用 repeat-y，重复铺满整个背景用 repeat。后面的 right bottom;是指背景图片从右下角开始。

color 用于定义字体颜色。

text-align 用来定义层中的内容排列方式：center 居中；left 居左，right 居右。

line-height 定义行高，120px 是指绝对高度为 120 个像素，也可以写作：line-height:150%，表示是标准高度的 150%。

width 是定义层的宽度，可以采用固定值，也可以采用百分比。这个宽度仅仅指内容的宽度，不包含 margin，border 和 padding 的宽度。

2. 盒子模型

自从 1996 年 CSS1 的推出，W3C 组织就建议把所有网页上的对象都放在一个盒(box)中，设计师可以通过创建定义来控制这个盒的属性，这些对象包括段落、列表、标题、图片以及层。盒子模型主要定义 4 个区域：内容(content)、边框距(padding)、边界(border)和边距(margin)。前面【例 7-5】中定义的层就是一个典型的盒。对于初学者，经常会搞不清楚 margin，background-color，background-image，padding，content，border 之间的层次、关系和相互影响。这里给出一个盒子模型图，如图 7-17 所示，方便初学者理解和使用。

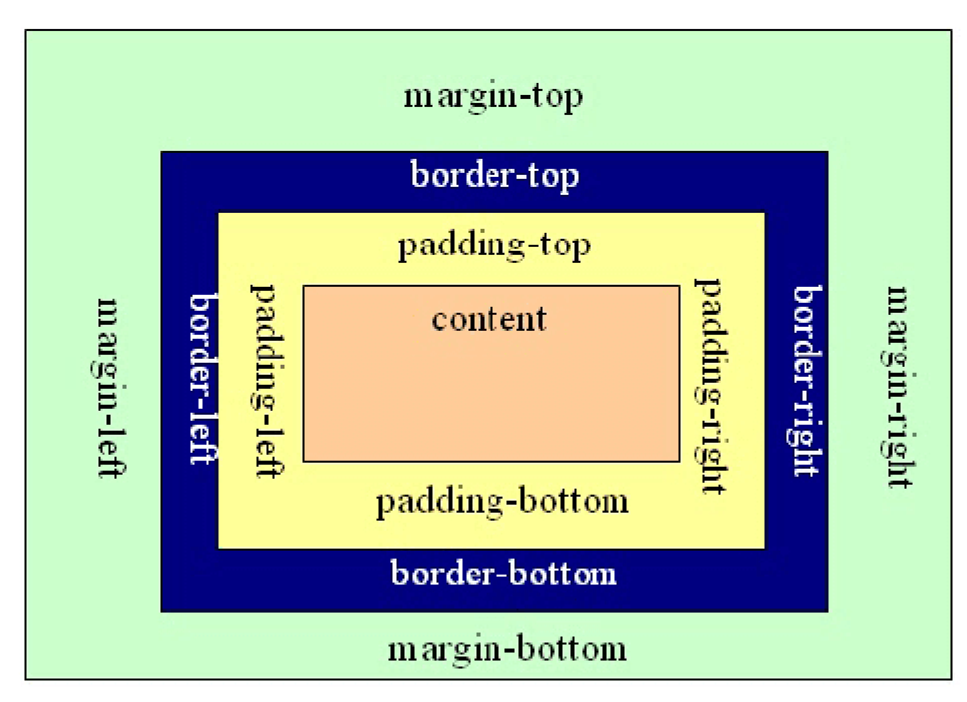


图 7-17　盒子模型

理解盒子模型就可以理解层与层之间定位的关系以及层内部的表达样式。其中，margin 属性负责层与层之间的距离，padding 属性负责内容和边框之间的距离。下面这段代码可以帮助读者进一步理解盒子模型的含义。

```
<head runat="server">
<title></title>
<style>
#sample2
{
background-color: #FFFF00;
border-style: solid;
padding-bottom: 25px;
margin-bottom: 50px;
width: 60%;
}
</style>
</head>
<body>
<form id="form1" runat="server">
```

<div id="sample2">W3C 组织就建议把所有网页上的对象都放在一个盒(box)中，设计师可以通过创建定义来控制这个盒的属性，这些对象包括段落、列表、标题、图片以及层</div>。

```
<p>这是下一段</p>
</form>
</body>
```

运行这段代码，效果如图 7-18 所示。

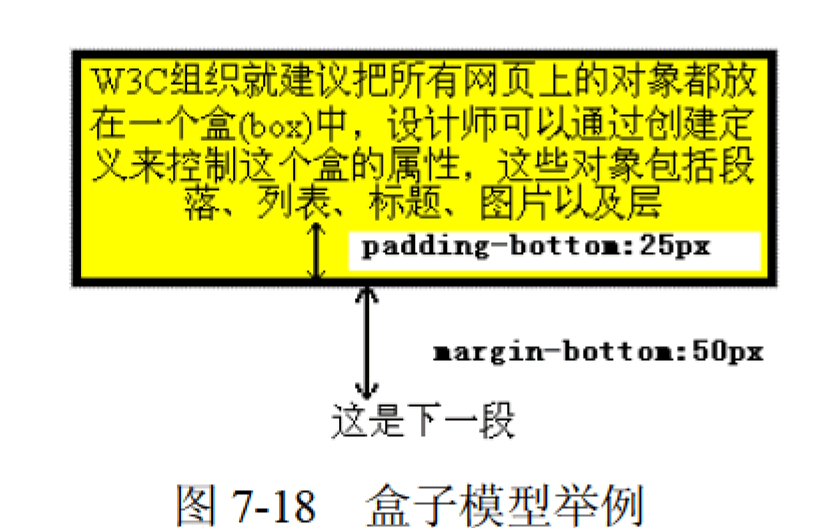


图 7-18　盒子模型举例

3. 层的定位

在一个页面中定义多个层，会发现这些层自动排列在不同的行，而要真正实现左右排列，就要加入新的属性——float(浮动属性)。float 浮动属性是 DIV 和 CSS 布局中的一个非常重要的属性。大部分的 DIV 布局都是通过 float 的控制来实现的。具体参数如下。

- float:none 用于设置是否浮动。
- float:left 用于表示对象向左浮动。
- float:right 用于表示对象向右浮动。

【例 7-6】下面通过一个左右分栏布局的例子来说明 float 的用法，该布局包含 2 个层且左右排列，这是最常用的布局结构之一，其效果如图 7-19 所示。

要想实现这样的效果，必须使用 float 属性，代码如下。

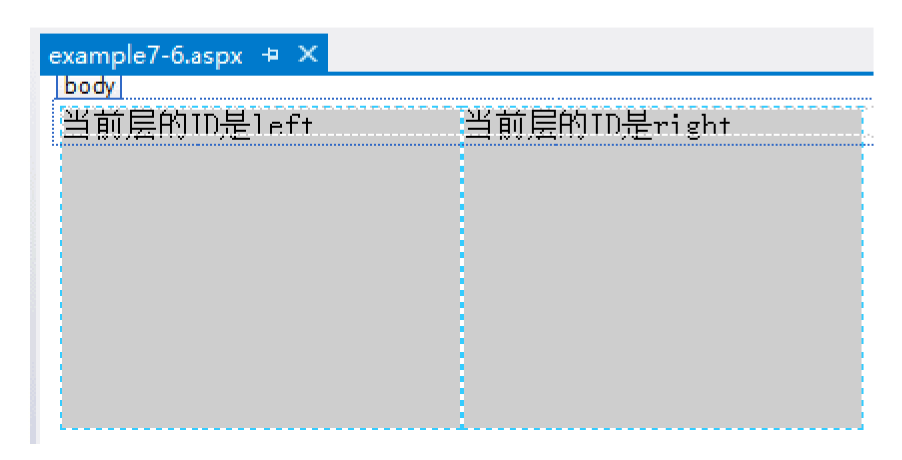


图 7-19　左右分栏效果

```
<head runat="server">
   <title></title>
    <style>
       #left,#right
       {
            width:200px;
            height:160px;
            background-color:#cecece;
            border:1px dashed #33ccff;
       }
       #left{float:left;}
       #right{float:left;}
    </style>
</head>
<body>
   <form id="form1" runat="server">
      <div id="left">当前层的 ID 是 left</div>
      <div id="right">当前层的 ID 是 right</div>
   </form>
</body>
```

读者可以尝试去掉#left{float:left}和#right{float:left}来看看会变成什么效果。当然，也可以把 float 的属性值改为 right 来看看会变成什么效果。

要想实现两列中左列宽度固定而右列宽度自适应窗口大小的效果，可以将【例 7-6】代码中的样式进行如下修改。

```
<style>
#left,#right{
      background-color:#cecece;
      border:1px solid #33ccff;
      height:400px;
}
#left{
width:180px;
      float:left;
}
</style>
```

这样一来，左边的层将呈现出 180px 的宽度。而右边的层则根据浏览器窗口大小来自动适应。

还有一种左右上下分栏的样式也是非常常见的，其效果如图 7-20 所示。

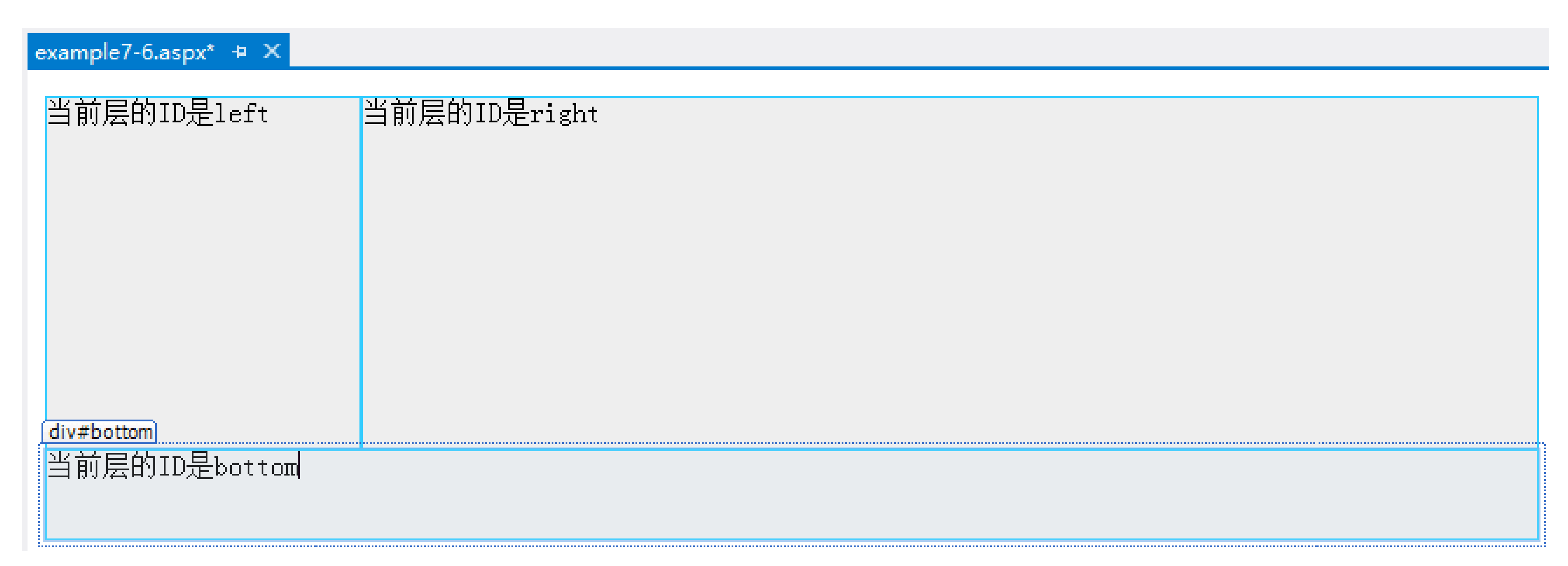


图 7-20　左右上下分栏

制作这种效果时需要将下面的层样式中添加 clear 属性，代码如下。

```
<head runat="server">
    <style>
#left,#right{background-color:#eeeeee;border:1px solid #33ccff;height:200px; }
#left{width:180px; float:left; }
#bottom{ background-color:#eeeeee; border:1px solid #33ccff; height:50px; clear:both; }
    </style>
</head>
<body>
  <form id="form1" runat="server">
    <div id="left">当前层的 ID 是 left</div>
    <div id="right">当前层的 ID 是 right</div>
    <div id="bottom">当前层的 ID 是 bottom</div>
  </form>
</body>
```

注意：

在 IE 浏览器中，即使不定义 clear 属性为 both，依然能够按照预期的效果显示下面的层对象。

4. 利用 DIV 和 CSS 实现页面布局

通过前面的介绍，可以知道 DIV 只是一个区域标识，划定了一个区域，实现样式需要借助于 CSS。这样的分离，使得 DIV 的最终效果是由 CSS 来编写的。CSS 可以实现左右分栏，也可以实现上下分栏，而表格则没有这么大的灵活性。CSS 与 DIV 的无关性，决定了 DIV 在设计上有极大的伸缩性，而不拘泥于单元格固定的模式束缚。因此，实现网页布局，通常是先在网页中将内容用 DIV 标记出来，然后再用 CSS 来编写样式。

采用 DIV 和 CSS 布局之前，首先要分析网页有哪些内容块，以及每个内容块的含义，这就是所谓的网页结构。通常情况下页面结构包含以下几块。

(1) 标题区(header)，用来显示网站的标志和站点名称等。

(2) 导航区(navigation)，用来表示网页的结构关系，如站点导航，通常放置主菜单。

(3) 主功能区(content)，用来显示网站的主题内容，如商品展示、公司介绍等。

(4) 页脚(footer)，用来显示网站的版权和有关法律声明等。

通常采用 DIV 元素来将这些结构先定义出来，示例代码如下。

```
<div id="header"></div>
<div id="globalnav"></div>
<div id="content"></div>
<div id="footer"></div>
```

现在还没有开始布局，这只是网页的结构，每一个内容块都可以放在页面上任何地方，放好以后就可以指定每个块的颜色、字体、边框、背景以及对齐属性等。本节将通过设计一个实际的网页，让初学者真正掌握利用 DIV 和 CSS 控制网页布局的技巧。

【例 7-7】设计一个如图 7-21 所示的某个网页的显示效果。

① 定义网站主页的结构，根据布局，可以将网页结构分为以下几个部分。

```
<body id="home">
<div id="top"></div>
<div id="header">
    <div id="logoarea">
        <div id="logo"></div>
```

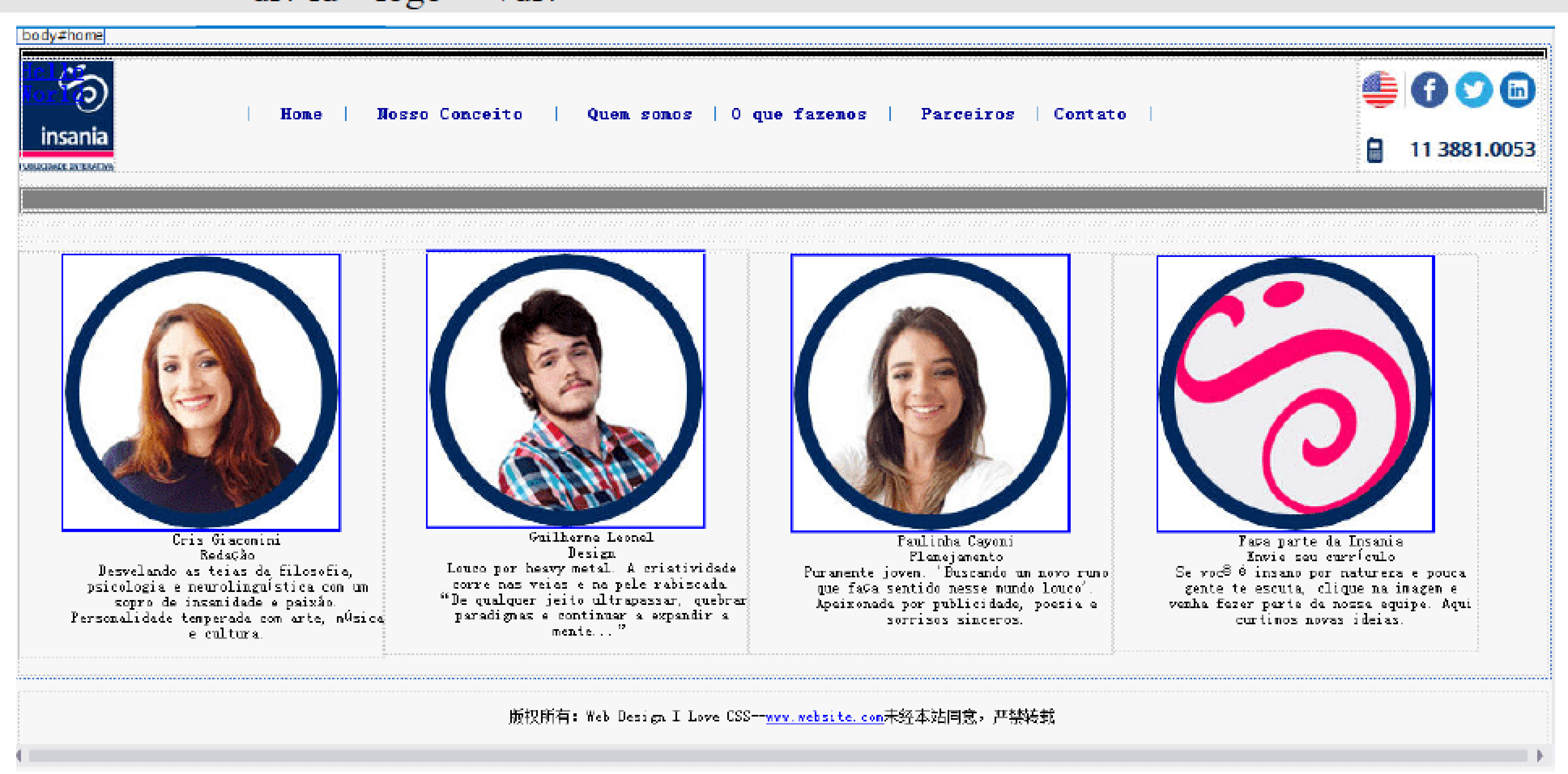


图 7-21　页面效果

```
            <div id="menu"></div>
            <div id="friend"></div>
    </div>
    <div id="banner"></div>
</div>
<div id="main"></div>
<div id="footer"></div>
</body>
```

② 制作如图 7-22 所示的图片。

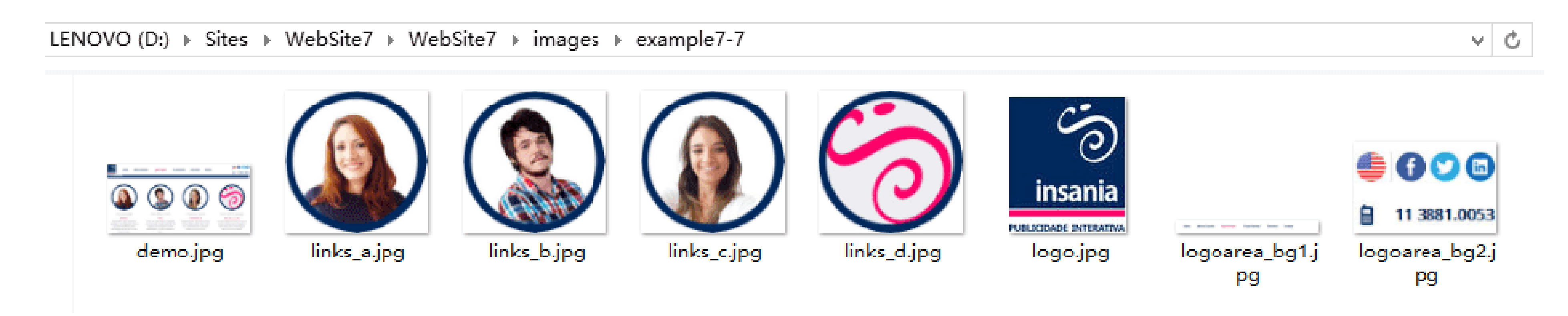


图 7-22　制作的图片

③ 在项目中添加一个名为 StyleSheet7-7.css 的样式表文件，然后利用样式表生成器在 body 中定义整个网页的默认字体以及网页对齐方式，代码如下。

```
body
{
    font-size: 12px;
    font-family: 宋体;
    text-align: center;
    background-color: #F7F7F7;
}
```

④ 在 StyleSheet7-7.css 文件中，定义整个网页的边距、宽度，代码如下。

```
#maindiv
{
    width: 1173px;
    margin-top: 0px;
    margin-left:auto;
    margin-right:auto;
}
```

⑤ 在页面中添加对样式表的引用，代码如下。

```
<head runat="server">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>某公司网站首页</title>
<link href="StyleSheet7-7.css" rel="stylesheet" type="text/css" />
</head>
```

⑥ 下面来定义页面的 top 层和 header 层。首先在 StyleSheet7-7.css 文件中，定义#top 的 样式，代码如下。

```
#top
{
    width:1173px;
    height:8px;
    background:#000000;
}
```

定义#header 的样式，代码如下。

```
#header
{
    width: 1173px;
```

```
        height: 115px;
        text-align: right;
        clip: rect(60px auto auto auto);
    }
```

在页面中定义 logo 层，代码如下。

```
    #header #logo
    {
        width:73px;
        height:87px;
        text-align:left;
        background-image:url(images/example7-7/logo.jpg);
        float:left;
    }
```

定义#menu 的样式，代码如下。

```
    #menu
    {
        width: 955px;
        height: 87px;
        color: #3381da;
        text-align:center;
        float:left;
    }
    #menu a, #menu a:link, #menu a:visited, #menu a:active
    {
        text-decoration: none;
        color: #000099;
        font-size: larger;
        font-weight: 700;
    }
    #menu a:hover
    {
        color: #ff0000;
        text-decoration: underline;
    }
```

定义#friend 层，代码如下。

```
    #header #friend
    {
        width:140px;
        height:87px;
        float:left;
        background-image:url(images/example7-7/logoarea_bg2.jpg);
    }
```

保存样式表文件，切换到页面的【设计】视图，添加超链接，得到如下代码：

```
    <div id="menu">
     <br />   <br />   <br /> |  
```

```
        <asp:HyperLink ID="HyperLink2" runat="server"
NavigateUrl="#">Home</asp:HyperLink>  |  
      <asp:HyperLink ID="HyperLink3" runat="server" NavigateUrl="#">Nosso Conceito
</asp:HyperLink>
        |  
      <asp:HyperLink ID="HyperLink4" runat="server" NavigateUrl="#">Quem somos</asp:HyperLink>
        |
      <asp:HyperLink ID="HyperLink5" runat="server" NavigateUrl="#">O que fazemos</asp:HyperLink>
        |  
      <asp:HyperLink ID="HyperLink6" runat="server" NavigateUrl="#">Parceiros</asp:HyperLink>
        |
      <asp:HyperLink ID="HyperLink7" runat="server" NavigateUrl="#">Contato</asp:HyperLink>
        |        
      </div>
```

此时页面设计效果如图 7-23 所示。

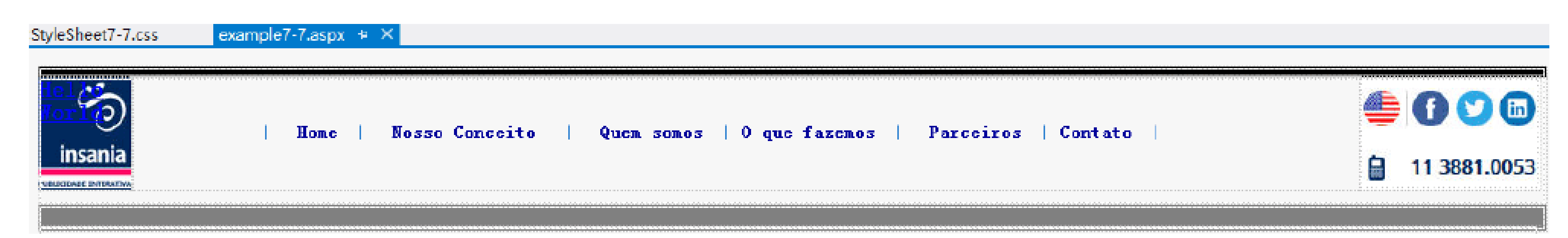


图 7-23　步骤⑥页面设计效果

⑦ 接着定义 banner 层，代码如下。

```
    #banner
    {
        width:1173px;
        height:20px;
        float:left;
        background:#808080;
    }
```

然后切换到页面的【设计】视图，编辑 main 层代码如下。

```
        <div id="main">
         <div id="mainarea1">
              <dl>
              <dt><a href="#"><img src="images/example7-7/links_a.jpg" alt="" title=""/></a></dt>
              <dd>Cris Giacomini<a href="#" title=""></a></dd>
              <dd>Redação<a href="#" title=""></a></dd>
              <dd>Desvelando as teias da filosofia, psicologia e neurolinguística com um sopro de
insanidade e paixão. Personalidade temperada com arte, música e cultura.<a href="#" title=""></a></dd>
              </dl>

         </div>
          <div id="mainarea2">
               <dl>
              <dt><a href="#"><img src="images/example7-7/links_b.jpg" alt="" title=""/></a></dt>
              <dd>Guilherme Leonel<a href="#" title=""></a></dd>
              <dd>Design<a href="#" title=""></a></dd>
              <dd>Louco por heavy metal. A criatividade corre nas veias e na pele rabiscada. “De qualquer
```

```
jeito ultrapassar, quebrar paradigmas e continuar a expandir a mente...”<a href="#" title=""></a></dd>
                    </dl>
            </div>
              <div id="mainarea3">
                    <dl>
                    <dt><a href="#"><img src="images/example7-7/links_c.jpg" alt="" title=""/></a></dt>
                    <dd>Paulinha Cayoni<a href="#" title=""></a></dd>
                    <dd>Planejamento<a href="#" title=""></a></dd>
                    <dd>Puramente jovem. 'Buscando um novo rumo que faça sentido nesse mundo louco'.
Apaixonada por publicidade, poesia e sorrisos sinceros.<a href="#" title=""></a></dd>
                     </dl>
              </div>
                <div id="mainarea4">
                        <dl>
                    <dt><a href="#"><img src="images/example7-7/links_d.jpg" alt="" title=""/></a></dt>
                    <dd>Faça parte da Insania<a href="#" title=""></a></dd>
                    <dd>Envie seu currículo<a href="#" title=""></a></dd>
                    <dd>Se você é insano por natureza e pouca gente te escuta, clique na imagem e venha fazer
parte da nossa equipe.
        Aqui curtimos novas ideias.<a href="#" title=""></a></dd>
                    </dl>
            </div>
        </div>
```

在 CSS 样式文件中定义#main，代码如下。

```
        #main #mainarea1
        dl{
                width:280px;
                height:312px;
                text-align:center;
                float:left;
                margin-top: 34px;
        }
        #main #mainarea2
        dl{
                width:280px;
                height:311px;
                text-align:center;
                float:left;
                margin-top: 20px;
        }
        #main #mainarea3
        dl{
                width:280px;
                height:308px;
                float:left;
                text-align:center;
                margin-top: 11px;
                margin-left: 0px;
        }
        #main #mainarea4
```

```
dl{
    width:280px;
    height:305px;
    text-align:center;
     margin-top: 0px;
}
```

此时页面效果如图 7-24 所示。

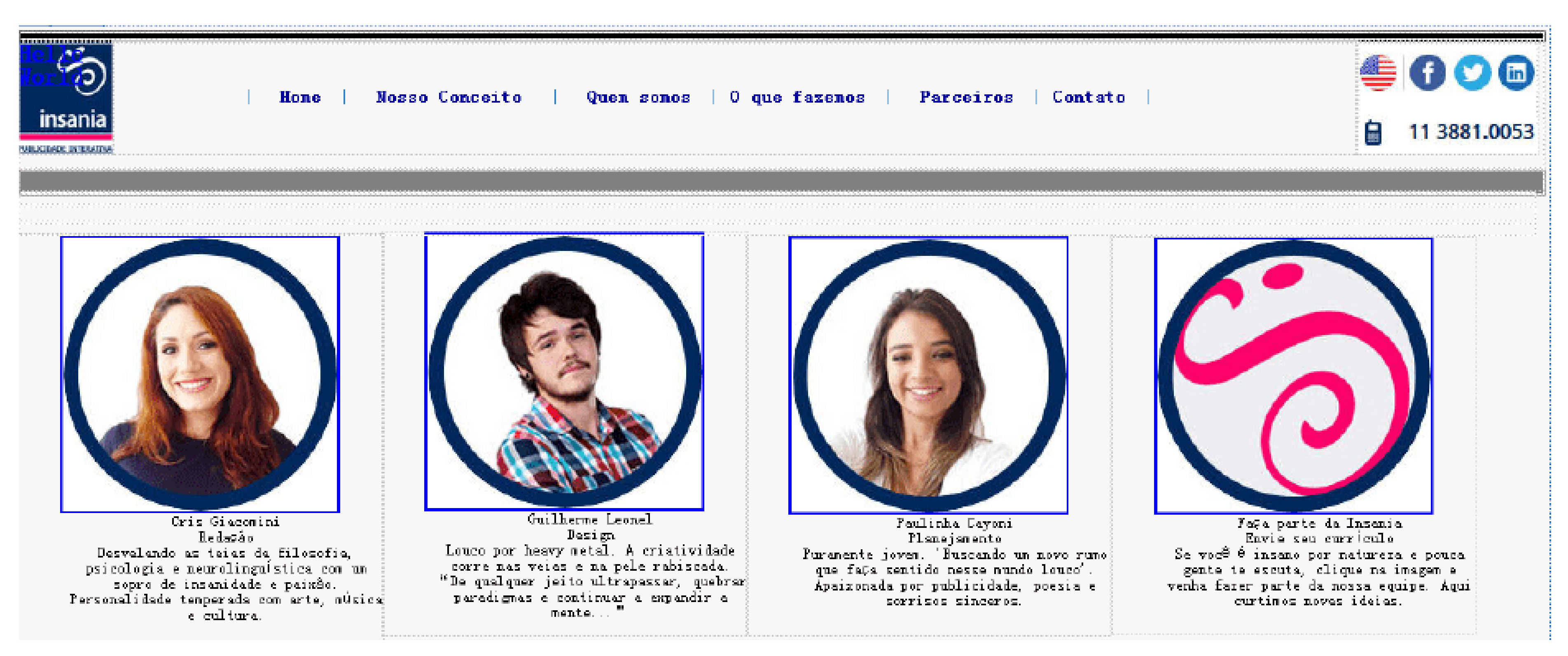


图 7-24　定义 main 层之后的页面效果

定义 footer 层，代码如下。

```
<div id="footer">
    <br />
    版权所有：Web Design I Love CSS--<a href="#" title="">www.website.com</a>未经本站同意，
严禁转载
</div>
```

在 CSS 文件中定义#footer，代码如下。

```
#footer
{
    width:1173px;
    height:50px;
    margin-top:8px;
}
```

保存样式表文件，切换到页面安的【设计】视图，显示如图 7-21 所示的效果。

7.3　主题

网站的美观主要涉及页面和控件的样式属性。在 ASP.NET 应用程序中，可以利用 CSS 控制各元素的样式以及部分服务器控件的样式，但是，有些服务器控件的属性无法通过 CSS 进行控制。为了解决这个问题，ASP.NET 提供了一种称为“主题”的新方式，它可以保持网站外观的一致性和独立性，同时使页面的样式控制更加灵活方便。例如，动态

实现不同用户界面的切换。

7.3.1　主题概述

主题是指页面和控件外观属性设置的集合。主题由一个文件组构成，包括皮肤文件(扩展名为.skin)、级联样式表文件(扩展名为.css)、图片和其他资源等，一个主题至少包含一个皮肤文件。

主题分为两大类型，一类是应用程序主题，另一类是全局主题。

应用程序主题是指保存在Web应用程序的App_Themes文件夹下的一个或多个主题文件夹，主题的名称就是文件夹的名称。

全局主题是指保存在服务器上，根据不同的服务器配置决定的，能够对服务器上所有Web应用程序起作用的主题文件夹。

一般情况下，很少用到全局主题，而本书所讲的主题仅指应用程序主题，即保存在应用程序中App_Themes文件夹下的主题文件夹，简称主题。

打开一个Web应用程序，在【解决方案资源管理器】中，右击项目名，选择【添加】|【添加ASP.NET文件夹】|【主题】命令，系统自动生成App_Themes文件夹，并在该文件夹下生成一个默认名为“主题1”的文件夹。在App_Themes文件夹中可以创建多个主题，方法相同。

1. 皮肤文件

皮肤文件是主题的核心文件，也称为外观文件，专门用于定义服务器控件的外观。在主题中可以包含一个或多个皮肤文件，后缀名为.skin。

在控件皮肤设置中，只能包含主题的属性定义，如样式属性、模板属性、数据绑定表达式等，不能包含控件的ID，如Label控件的皮肤设置代码如下。

```
<asp:Label runat="server" BackColor="Blue" Font-Names="Arial Narrow" />
```

这样一旦将该皮肤应用到Web页面中，则所有的Label控件都将显示皮肤所设置的样式。

右击某一个“主题”文件夹，选择【添加】|【新建项】命令，在弹出对话框中选择SkinFile，并在【名称】文本框中修改皮肤文件名，单击【添加】按钮即可添加一个皮肤文件。同样的方法可以添加多个皮肤文件。

2. 级联样式表文件

主题中可以包含一个或多个CSS文件，一旦CSS文件被放在主题中，则应用时无须再在页面中指定CSS文件链接，而是通过设置页面或网站所使用的主题即可，当主题得到应用时，主题中的CSS文件会自动应用到页面中。

右击某一个“主题1”文件夹，选择【添加】|【新建项】命令，在弹出的对话框中选择SkinFile，并在【名称】文本框中修改样式表文件名，单击【添加】按钮即可添加一个样式表文件。同样的方法可以添加多个样式表文件。

如图 7-25 所示创建了两个主题，分别是“主题 1”和“主题 2”，其中分别包含了两个皮肤文件。

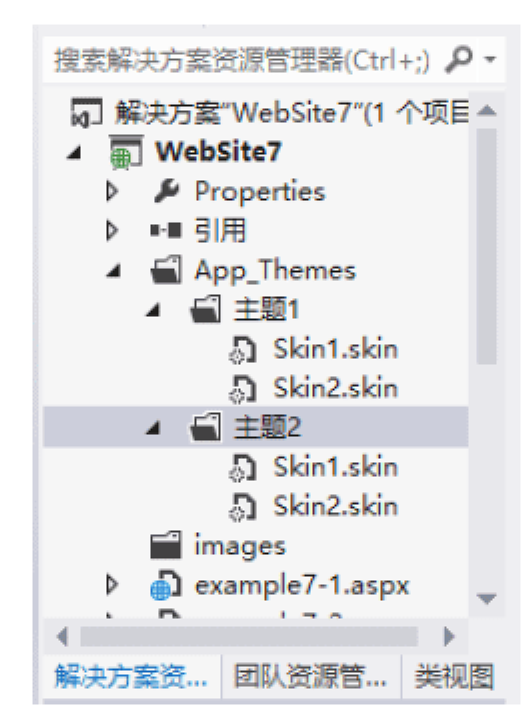


图 7-25　定义多个主题

7.3.2　在主题中定义外观

ASP.NET 使得将预定义的主题应用于页面或创建唯一主题的操作变得更容易。下面通过一个简单的例子来说明定义外观的方法。

【例 7-8】创建一个包含简单外观的主题，这些外观用于定义控件的外观。

(1) 在 VS 中，右击网站名 WebSite7，选择【添加 ASP.NET 文件夹】|【主题】命令，将创建名为 App_Themes 的文件夹和名为“主题 1”的子文件夹。将【主题 1】文件夹重命名为 Theme1，此文件夹名将成为创建的主题的名称。

(2) 右击 Theme1 文件夹，选择【添加】|【新建项】命令，添加一个外观文件 sampleSkin.skin。在 sampleSkin.skin 文件中，按如下代码所示的方法添加外观定义。

```
<asp:Label runat="server" ForeColor="red" Font-Size="14pt" Font-Names="Verdana" />
<asp:button runat="server" Borderstyle="Solid" Borderwidth="2px" Bordercolor="Blue"
Backcolor="yellow"/>
```

外观定义与创建控件的语法类似，不同之处在于，定义只包括影响控件外观的设置，不包括 控件的 ID 属性设置。

(3) 保存该外观文件。

(4) 新建一个网页 example7-8.aspx，切换到【设计】视图中，添加 1 个标签控件和 1 个按钮控件，如图 7-26 所示。

(5) 在【属性】面板中选择 Document 元素，设置 Theme 属性的值为 Theme1，切换到【源】视图中，会发现代码第 1 行的@ Page 指令中添加了下面的属性。

```
<%@ Page … Theme="Theme1"%>
```

(6) 保存文件，按 Ctrl+F5 组合键执行该页面查看设置效果，如图 7-27 所示。

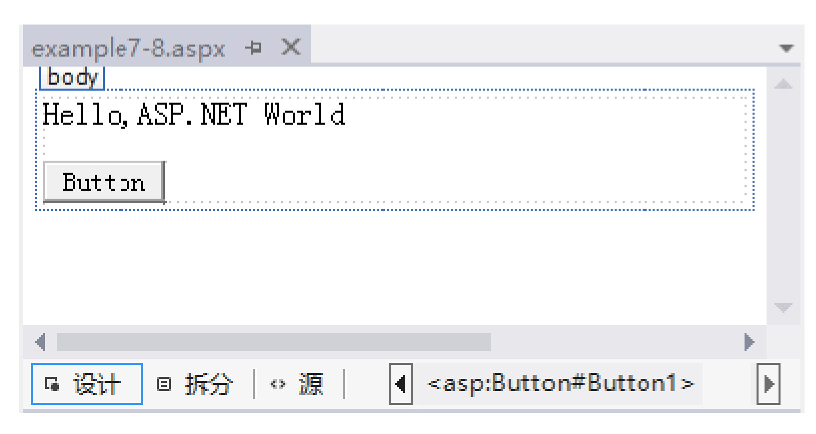


图 7-26　设置外观前

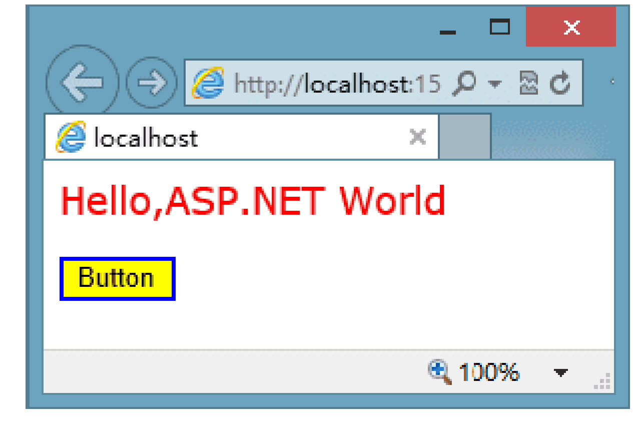


图 7-27　设置外观后

在该网页文件中，变更所使用的主题，然后再按 Ctrl+F5 组合键运行程序，控件将再次更改外观为相应的主题。

在皮肤文件中，系统没有提供控件属性设置的智能提示功能。因此，一般不直接在皮肤文件中定义控件外观，而是先在页面中设置控件的属性，然后再将自动生成的代码复制

到外观文件中。因此，【例 7-8】也可以用下面的方法来实现。

【例 7-8】定义并应用皮肤文件。

(1) 创建一个 Web 页面，添加相应的控件并设置其外观。

(2) 新建一个主题，将 Web 页面中相应控件的源代码复制到该主题的皮肤文件中，并去掉所有控件的 ID 属性。

(3) 在其他页面的【属性】面板中选择 Document 元素，设置 Theme 属性的值为相应的主题即可。

如果希望某些控件的外观和页面中具有相同类型的其他控件的外观不一样，则可以直接在皮肤文件中指定，即在.skin 文件中给指定控件添加一个 SkinID 属性。例如，在【例 7-9】中增加一个按钮，其外观定义成如下样式。

```
    <asp:Button runat="server" SkinID="GreenButton" Borderstyle="dotted" Borderwidth="2px"
Bordercolor="red" Backcolor="Green"/>
```

修改按钮控件的 SkinID 属性的值为 GreenButton。这样，新增加的按钮就和原来的按钮显示了不同的外观。

7.3.3 在主题中同时定义外观和样式表

【例 7-9】定义并应用了一个皮肤文件，实际上，在主题中还可以定义 CSS 文件。要想让定义的 CSS 文件起作用，需要在网页文件中设置 StyleSheetTheme 属性为定义的主题即可。

【例 7-10】演示如何在网页文件中同时使用皮肤文件和样式表文件。

(1) 在 VS 中，右击网站名 WebSite7，选择【添加】|【添加 ASP.NET 文件夹】|【主题】命令，将创建名为 App_Themes 的文件夹和名为“主题 1”的子文件夹。将【主题 1】文件夹重命名为 Theme2。此文件夹名将成为创建的主题的名称。

(2) 右击 Theme2 文件夹，选择【添加】|【新建项】命令，添加一个新的外观文件，然后将该文件命名为 Skin1.skin。在 Skin1.skin 文件中，添加网页文件中要用到的所有控件的外观定义，注意不能含有任何控件的 ID，外观代码如下。

```
<asp:Label runat="server" BackColor="#FFFFCC" BorderColor="#6600FF"
  BorderStyle="Solid" BorderWidth="4px" Font-Bold="True" Font-Names="华文彩云"
  Font-Size="XX-Large" ForeColor="#CC0099" style="text-align: center" Width="206px">
</asp:Label>
<asp:Button runat="server" BackColor="#3333CC" BorderColor="#000099"
Font-Bold="True" Font-Size="Medium" ForeColor="White"/>
<asp:TextBox runat="server" BackColor="#99FFCC" Columns="10"></asp:TextBox>
```

(3) 在主题 Theme2 文件夹下，添加一个 Stylesheet1.css 的样式文件，其代码如下。

```
.style1    /*  用于修饰表格  */
{
    width: 200px;
    border-collapse: collapse;
```

```
        border: 1px solid #800080;
    }
    .style2    /*  用于修饰单元格  */
    {
        font-family: 幼圆;
        font-size: large;
        font-weight: bold;
    }
```

(4) 新建一个 Web 页面 example7-9.aspx，切换到【设计】视图，添加表格和相应控件，其最终效果如图 7-28 所示。

修改当前页面的 Document 中属性 StyleSheetTheme 的值为 Theme2，即可看到引入皮肤和样式表文件后的最终显示效果，如图 7-29 所示。

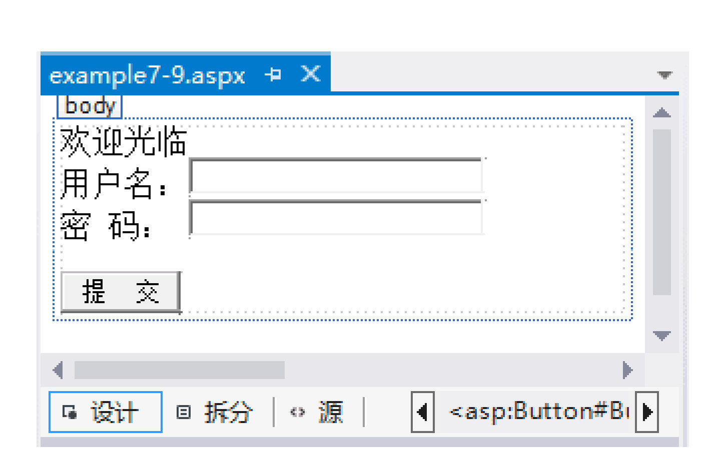


图 7-28　没有引入皮肤和样式前的效果

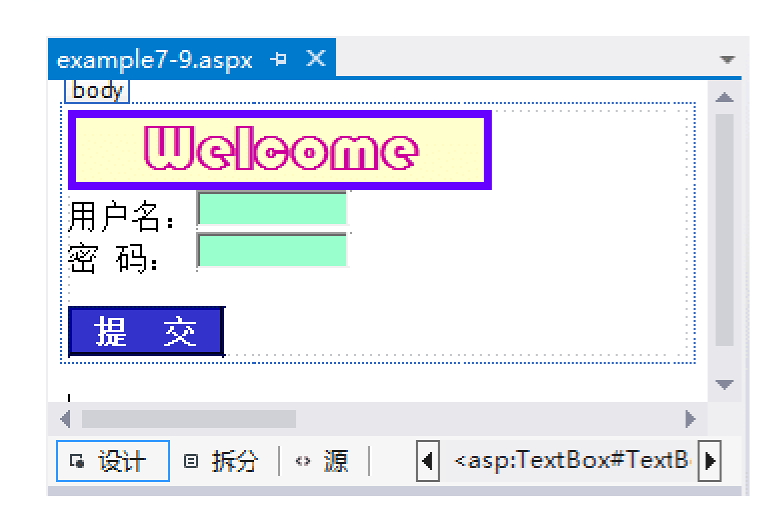


图 7-29　引入皮肤和样式后的效果

需要注意的是，创建了主题后，可以定制如何在应用程序中使用主题，操作方法为：将主题作为自定义主题与网页文件关联，或者将主题作为样式表主题与网页文件关联。样式表主题和自定义主题都使用相同的主题文件，但是样式表主题在网页文件的控件和属性中的优先级最低。在 ASP.NET 中，优先级的顺序从高到低如下。

(1) 主题设置，包括 Web.config 文件中设置的主题。

(2) 本地网页文件的样式属性。

(3) 样式表主题。

如果选择使用样式表主题，则在网页文件中本地声明的任何样式信息都将覆盖样式表主题的属性。同样，如果使用自定义主题，则主题的属性将覆盖本地网页文件中设置的任何样式内容，以及使用中的任何样式表主题中的任何内容。

7.3.4　利用主题实现换肤

前面介绍了如何指定页面主题的方法，但没有涉及后台代码。但在实际使用中，可能需要通过后台代码来控制主题的指定。例如，通过按钮来为页面指定不同的主题，以实现换肤功能，如 QQ 软件、浏览器的换肤功能。

下面的【例 7-11】使用设计器来设置控件外观属性，然后将生成的代码复制到外观文件，通过基于现有控件来创建自定义主题，这是非常简单的一种方法。

【例 7-11】利用主题实现换肤功能。

(1) 在 VS 中，右击网站名 WebSite7，选择【添加】|【添加 ASP.NET 文件夹】|【主题】命令，将在 App_Themes 的文件夹下创建名为“主题 1”的子文件夹。将【主题 1】文

件夹重命名为 Theme3。此文件夹名将成为创建的第一个主题的名称。同样的方法创建第二个主题，重命名为 Theme4。

(2) 右击 Theme3 文件夹，选择【添加】|【新建项】命令，添加一个新的外观文件，重命名为 Skin1.skin。同样，在 Theme4 文件夹中添加一个外观文件，重命名为 Skin2.skin。

(3) 新建一个 Web 页面 example7-11.aspx，切换到【设计】视图，添加 2 个“日历”控件，设置第 1 个“日历”控件的自动套用格式为“专业型 1”，设置第 2 个“日历”控件的自动套用格式为“专业型 2”。切换到【源】视图，并复制这两个<asp:calendar>元素及其属性，分别粘贴到 Skin1.skin 文件中和 Skin2.skin 文件中。然后删除每一个<asp:calendar>元素中的 ID 属性。保存这两个皮肤文件。

(4) 切换到 example7-11.aspx 页面，将一个“日历”控件拖动到页上，不要设置该控件的任何属性；然后再添加两个“按钮”控件。效果如图 7-30 所示。



图 7-30　没有引入主题的日历

(5) 双击【专业型 1】按钮，为其 Click 事件创建一个处理程序，代码如下。

```
protected void Button1_Click(object sender, EventArgs e)
{
        Session["themepage"] = "Theme3";   // 将专业型 1 主题保存到 Session 中
        Response.Redirect(Request.Url.ToString());
}
```

(6) 双击【专业型 2】按钮，为其 Click 事件创建一个处理程序，代码如下。

```
protected void Button2_Click(object sender, EventArgs e)
{
        Session["themepage"] = "Theme4";    // 将专业型 2 主题保存到 Session 中
        Response.Redirect(Request.Url.ToString());
}
```

(7) 在类定义中再添加如下代码。

```
void Page_PreInit(object sender, EventArgs e)
{
     if (Session["themepage"] != null)
     {
          this.Theme = (String)Session["themepage"];
     }
     else
     {
          this.Theme = "Theme3";   // 默认加载专业型 1 主题
     }
}
```

(8) 测试运行效果。在 Default.aspx 页中按 Ctrl+F5 组合键运行此页面，此时会在浏览器中呈现，默认以专业型 1 主题显示日历，如图 7-31 所示。单击 【专业型 2】按钮，以

专业型 2 主题显示日历，如图 7-32 所示。单击不同的按钮就可以实现换肤的效果了。

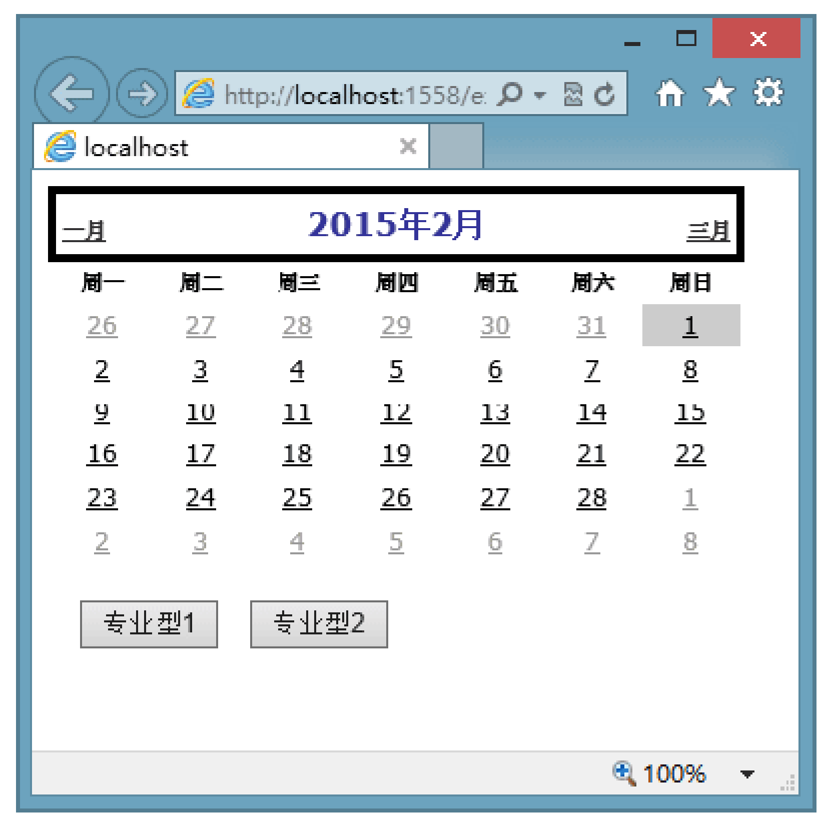



图 7-31　专业型 1 主题

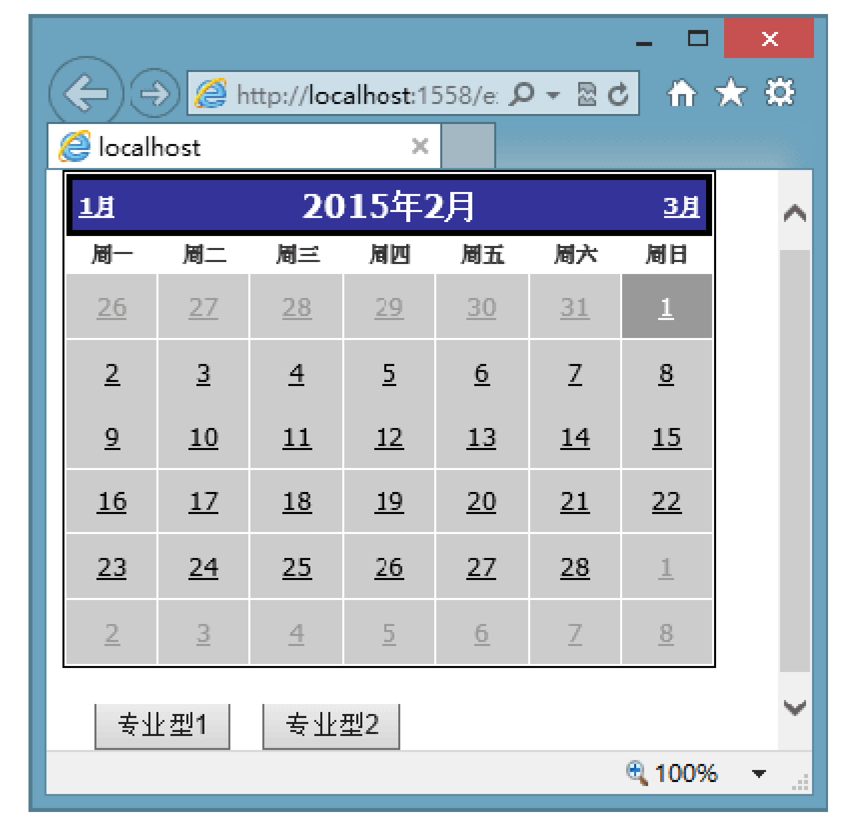


图 7-32　专业型 2 主题

前面的内容都是将主题应用于某一个页面文件，其实还可以将一个主题应用于整个网站。为网站设置主题的方法如下。

(1) 打开网站的配置文件 Web.config，在<system.web>元素内部添加<pages>元素。

(2) 将下列属性添加到<pages>元素中。

```
<pages theme="sampleTheme" />
```

(3) 保存并关闭 Web.config 文件。

(4) 打开所有包含主题的页面，并切换到【源】视图，从页声明中移除 theme="themeName" 属性。

此后，网站中所有页面都将使用 Web.config 文件中指定的主题显示。如果选择在某一个页面文件的声明中再指定一个主题名称，则该主题名称将覆盖 Web.config 文件中指定的任何主题。

7.4　母版页

在 Web 站点开发中，有很多元素，如站点标题、公共导航以及版权信息等，会出现在每一个页面中，这些元素的一致布局会让用户知道自己始终是在同一个站点中。虽然这些元素可以通过在 XHTML 中使用包含文件构建，但 ASP.NET 提供了更加健壮的母版页技术来实现。

7.4.1　母版页和内容页的概念

母版页是用于设置页面外观的模板，是一种特殊的 ASP.NET 网页文件，同样也具有其他 ASP.NET 文件的功能，如添加控件、设置样式等，只不过扩展名是.master。在母版页中，界面被分为公用区和可编辑区，公用区的设计方法与一般页面的设计方式相同，可编辑区用 ContentPlaceHolder 控件预留出来。

引用母版页的.aspx 页面称为内容页，在内容页中，母版页的 ContentPlaceHolder 控件预留的可编辑区会被自动替换为 Content 控件，开发人员只需要在 Content 控件区域中填充内容即可。在母版页中定义的其他标记将自动出现在引用该母版页的.aspx 页面中，母版页的部分以灰色显示，表示不能修改这些内容。

每一个母版页中可以包含一个或多个内容页。使用母版页可以统一管理和定义具有相同布局风格的页面，给网页设计和修改带来极大的方便。母版页的优点如下。

(1) 使用母版页可以集中处理页的通用功能，以便可以只在一个位置进行更新。

(2) 使用母版页可以方便地创建一组控件和代码，并将结果应用于一组新的页面。

(3) 通过允许控制占位符控件的呈现方式，母版页可以在细节上控制最终页的布局。

(4) 母版页提供一个对象模型，使用该对象模型可以从各个内容页自定义母版页。

在使用母版页时，母版页中使用的图片和超链接应尽量使用服务器端控件来实现，如 Image 和 HyperLink 控件，即使控件不需要服务器代码也是如此。这是因为将设计好的母版页或内容页移动到另一个文件夹时，如果使用的是服务器控件，即使不改变服务器控件的 URL，ASP.NET 也可以正确解析，并自动将其 URL 改为正确的位置。但是如果使用了普通 HTML 标记，那么 ASP.NET 将无法正确解析这些标记的 URL，从而导致图片不能显示和链接失败，不方便维护。

7.4.2　创建母版页和内容页

创建母版页的方法和创建一般页面的方法非常相似，区别是母版页无法单独在浏览器中查看，必须通过创建内容页才能浏览。下面的【例7-8】中，母版页包含一个标题、一个导航菜单和一个页脚，这些内容将在站点的每个页面中出现。在母版页中包含一个内容占位符，这是母版页中的一个可变区域，可以使用内容页中的信息来替换此区域。

【例 7-12】设计如图 7-33 所示的名为 Master1.Master 的母版页，然后设计两个引用母版页的内容页 Index.aspx 和 About.aspx，运行效果分别如图 7-34 和图 7-35 所示。

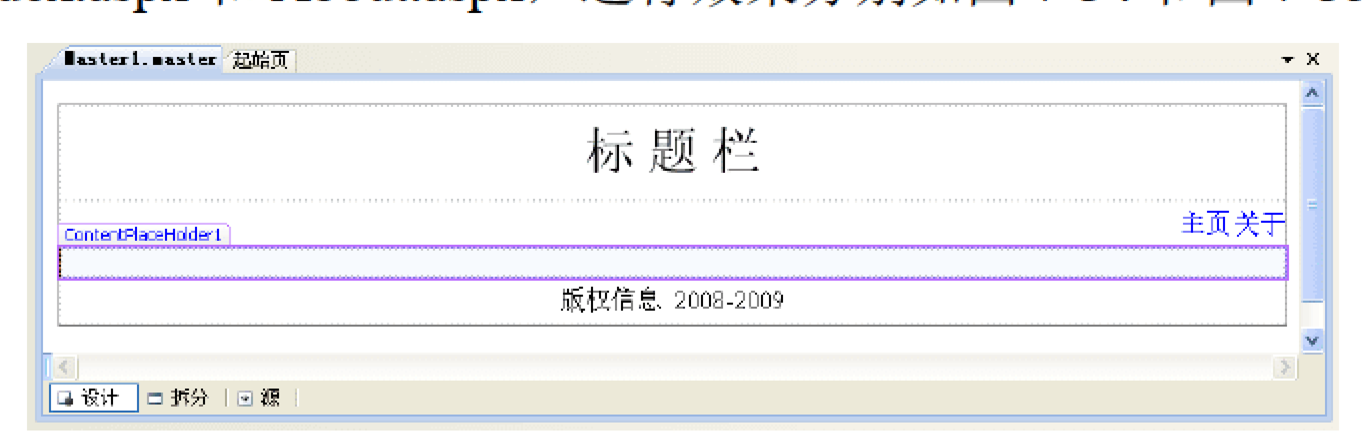


图 7-33　母版页布局

(1) 在 VS 的【解决方案资源管理器】中右击网站名称，然后单击【添加】|【新建项】命令，然后在弹出对话框中单击 MasterPage。在【名称】文本框中输入 Master1.Master，如图 7-36 所示，然后单击【添加】按钮，即会在【源】视图中打开新的母版页。

观察母版页的源代码，在页面的顶部是一个 @ Master 声明，而不是通常在 ASP.NET 页顶部看到的 @ Page 声明，指令如下。

```
    <%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Master1.master.cs"
Inherits="WebSite7.Master1" %>
```

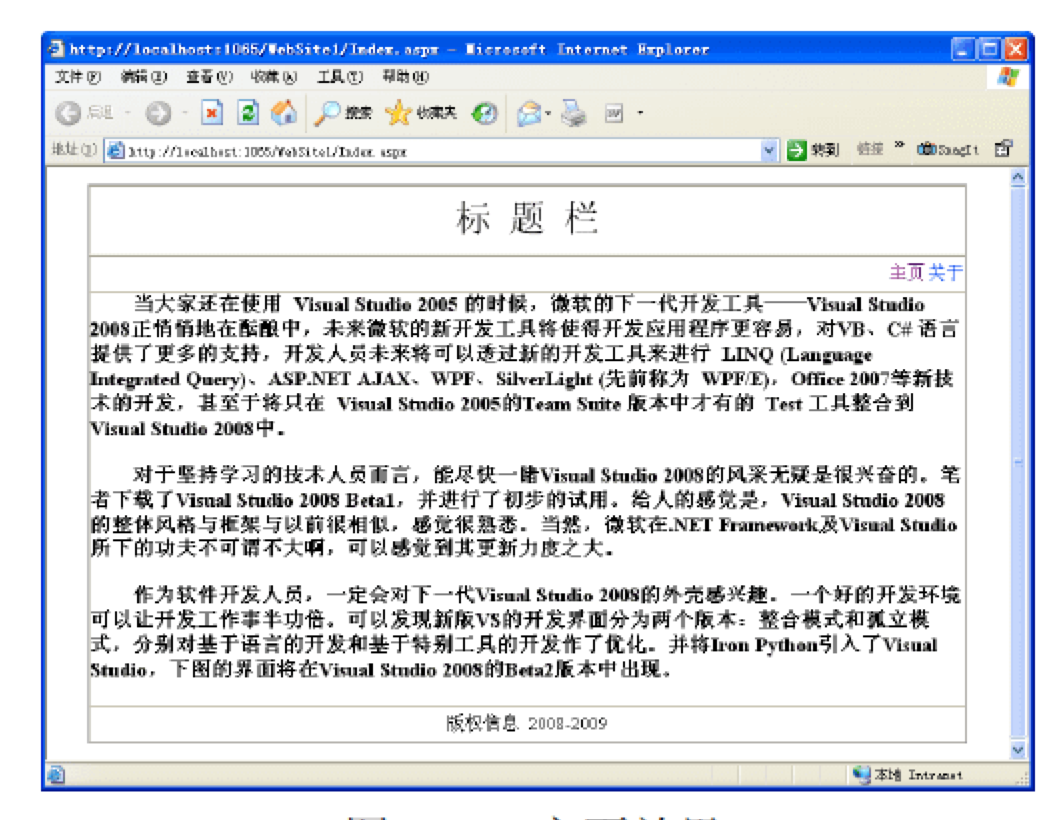


图 7-34　主页效果

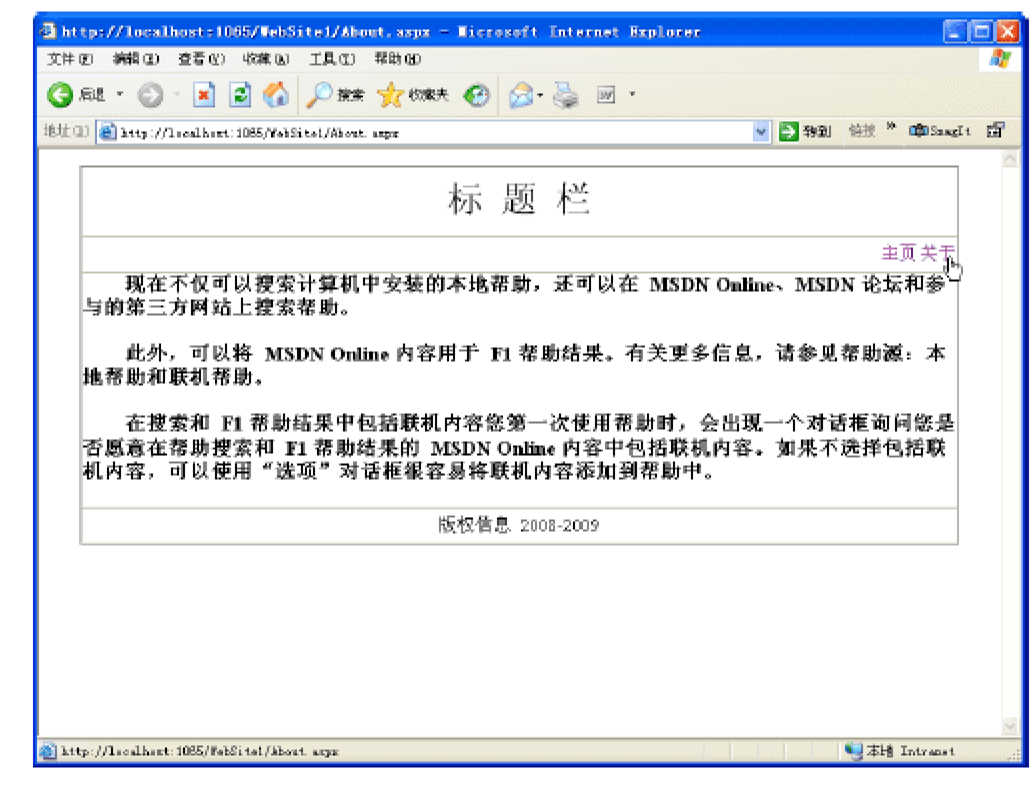


图 7-35　“关于”页效果

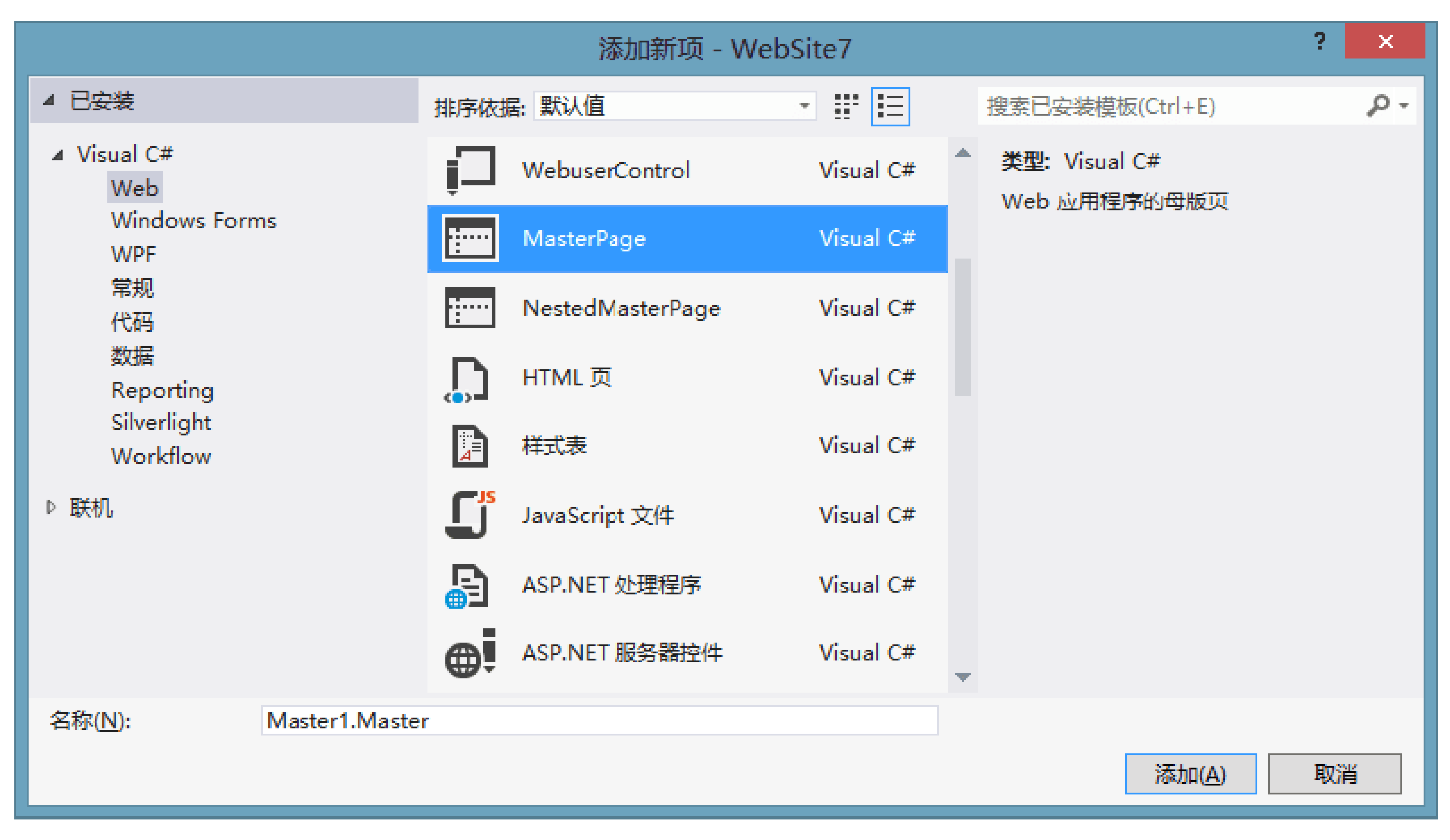


图 7-36　创建母版页

此外，页面的主体还包含一个 ContentPlaceHolder 控件，这是母版页中的一个区域，其中的可替换内容将在运行时由内容页合并。为了方便母版页的编辑，通常情况下先将 ContentPlaceHolder 控件删除，母版页编辑完成后再放置 ContentPlaceHolder 控件。

(2) 切换到【设计】视图，删除 ContentPlaceHolder 控件，然后单击页面中的层，插入一个 4 行 1 列的表格，边框设置为 1，表格的 width 设置为 780 像素。

(3) 插入表格后，可以将内容添加到母版页，此内容将在所有页面上显示。例如，可以在表格第 1 行添加“标题栏”，第 2 行添加 1 个 Menu 控件，第 3 行添加 1 个 ContentPlaceHolder 控件，控件的 ID 属性为 ContentPlaceHolder1，可以修改这个名字，第 4 行添加“版权信息”。其中 Menu 控件的设置内容如下。

- 将 Menu 控件的 Orientation 属性设置为 Horizontal。
- 单击 Menu 控件上的智能标记，选择【编辑菜单项】命令，然后在【菜单项编辑器】对话框中单击【添加根项】命令图标两次，添加两个菜单项。
- 单击第一个节点，将 Text 设置为“主页”，将 NavigateUrl 设置为 Index.aspx。
- 单击第二个节点，将 Text 设置为“关于”，将 NavigateUrl 设置为 About.aspx。

然后就要为母版页添加两个带有内容的页面。第一个是主页，第二个是“关于”页面。

(4) 在【解决方案资源管理器】中右击 Master1.master，然后选择【添加】|【内容页】命令，新建并打开 WebForm1.aspx，重命名为 Index.aspx。该页面包含一个 @ Page 指令，此指令将当前页附加到带有 Master1 属性的选定母版页，如下代码所示。

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Master1.Master" AutoEventWireup="true"
CodeBehind="Index.aspx.cs" Inherits="WebSite7.WebForm1" %>
    <asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
    </asp:Content>
```

(5) 切换到【设计】视图。母版页中的 ContentPlaceHolder 控件在新的内容页中显示为 Content 控件。而其他的母版页内容显示为浅灰色，表示在编辑内容页时不能更改这些内容。在与母版页上的 ContentPlaceHolder1 匹配的 Content 控件中，输入主页要显示的内容，然后选择文本，通过从【工具箱】上的【块格式】下拉列表中选择【标题 2】，保存页面。

(6) 使用同样的方法创建“关于”内容页，名字为 About.aspx。

(7) 设置 Index.aspx 为起始页，按 Ctrl+F5 组合键运行并测试这个站点。ASP.NET 将 Index.aspx 页的内容与 Master1.master 页的布局合并到单个页面，并在浏览器中显示产生的页面。需要注意的是，此页的 URL 为 Index.aspx，浏览器中是不存在对母版页的引用的。单击“关于”链接，显示 About.aspx 页，它也是和 Master1.master 页合并的结果。

7.4.3　从内容页访问母版页的成员

利用内容页的后台代码可以引用母版页上的成员，包括母版页上的任何公共属性或方法以及任何控件。要实现内容页对母版页中定义的属性或方法进行访问，则该属性或方法必须声明为公共成员(public)，也可以对母版页动态地进行访问。

1. 访问母版页的公共成员

要想在内容页中访问母版页上的属性，必须在母版页上先创建一个属性，创建方法如下：

(1) 切换到或打开 Master1.master。在【解决方案资源管理器】中右击 Master1.master，然后单击【查看代码】命令，打开代码编辑器。

(2) 在类定义中输入以下代码。

```
public String WebSiteName
{
    get { return (String) ViewState["WebsiteName"]; }
    set { ViewState["WebsiteName"] = value; }
}
```

此代码为母版页创建名为 WebSiteName 的属性。在视图状态中存储此值，以便在访问期间保持不变。

(3) 在类定义中再添加以下代码。

```
void Page_Init(Object sender, EventArgs e)
{
    this.WebSiteName = "母版页测试";
}
```

接下来修改内容页，通过引用 WebSiteName 属性，来使用母版页的公共成员。

(4) 切换到或打开 Index.aspx 页，并切换到该页的【源】视图。在页面顶部的 @ Page 指令下，添加下面的 @ MasterType 指令。

```
<%@ MasterType virtualpath="~/Master1.master" %>
```

此指令的作用是将内容页的 Master 属性绑定到 Master1.master 页。

(5) 切换到【设计】视图，在 Content 控件中，增加一行内容“该网站是关于”。

(6) 从【工具箱】中将 Label 控件拖动到 Content 控件上，并将其放置到静态文本的后面，使文本如下。

```
该网站是关于 [Label]
```

(7) 在【解决方案资源管理器】中右击 Index.aspx，然后单击【查看代码】打开代码编辑器。在类定义中添加下面的代码。

```
void Page_Load(Object sender, EventArgs e)
{
    Label1.Text = Master.WebSiteName;
}
```

(8) 测试内容页，切换到或打开 Index.aspx 页，然后按 Ctrl+F5 组合键运行页面。页面即会在浏览器中显示，其中文本为“该网站是关于母版页测试”。

(9) 修改母版页中属性的值，重新运行页面将看到新的属性值显示在页面中。

2. 动态访问母版页

有些时候，程序员希望能够动态更改母版页，即使用代码为内容页指定母版页。例如，允许用户根据喜好从几个布局中进行选择。

首先，要保证网站中有不少于两个母版页，然后创建按钮，以使用户能够在两个母版页之间进行选择切换。在下面的示例中，第二个母版页和第一个母版页非常相似，只是将标题栏和版权信息的字体稍作修改。具体操作步骤如下。

(1) 切换到或打开 Master1.master，在【工具箱】中将 LinkButton 按钮控件拖动到页面上，并将其放置在标题栏内容的右边。

(2) 将按钮控件的 Text 属性设置为“变换字体的样式”，如图 7-37 所示。

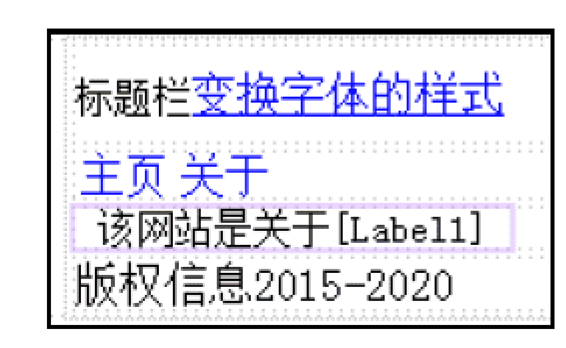


图 7-37　修改后的 Master1 的设计界面

(3) 双击此按钮，为其 Click 事件创建一个处理程序，然后添加以下代码。

```
void LinkButton1_Click(Object sender, EventArgs e)
{
    Session["masterpage"] = "Master2.master";
    Response.Redirect(Request.Url.ToString());
}
```

此代码将第二个母版页的文件名加载到一个持久的会话变量中，然后重新加载当前页。

(4) 创建第二个母版页，内容和 Master1.master 完全相同，只是字体效果不同，和 Master1 一样，添加 1 个 LinkButton 控件，并将其 Text 属性设置为“返回正常样式”。如图 7-38 所示。

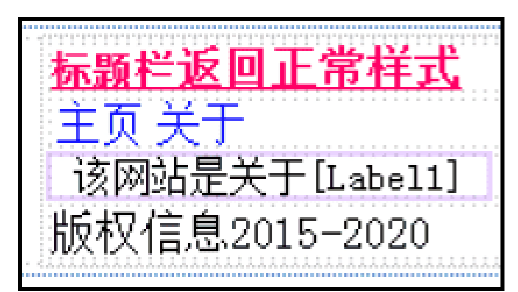


图 7-38　Master2 的效果

(5) 双击此按钮，为其 Click 事件创建一个处理程序，然后添加下面的代码。

```
void LinkButton1_Click(Object sender, EventArgs e)
{
    Session["masterpage"] = "Master1.master";
    Response.Redirect(Request.Url.ToString());
}
```

此代码与 Master1.master 页中按钮的代码类似，不同之处在于它加载第一个母版页。最后一步是在内容页中编写代码，此代码将动态加载用户选定的母版页。

(6) 切换到或打开 About.aspx 页。在【解决方案资源管理器】中右击 About.aspx，然后单击【查看代码】命令，打开代码编辑器。在类定义中添加如下代码。

```
void Page_PreInit(Object sender, EventArgs e)
{
    if(Session["masterpage"] != null)
    {
        this.MasterPageFile = (String) Session["masterpage"];
    }
}
```

此代码将当前页的 MasterPageFile 属性的值设置为会话变量中的值(如果有)。此代码必须在 Page_PreInit 处理程序中运行，因为必须建立母版页，使得页面可以创建其实例，然后可以进一步初始化。

(7) 测试动态母版页。在 About.aspx 页中按 Ctrl+F5 组合键运行此页面。此页即会在浏览器中显示，它与其默认母版页 Master1.master 合并。单击“变换字体的样式”链接，将会重新显示此页，但这一次它是与 Master2.master 合并。

7.5　本章小结

本章全面讲述了 ASP.NET 中的样式、主题和母版功能的知识及使用方法。首先介绍了介绍了 CSS 基础知识，内容包括 CSS 语言概述、CSS 样式引用的方法；然后介绍了页面布局知识，内容包括网页的基本布局方式、页面元素的定位、表格布局、DIV 和 CSS 布局等；接着讲解了主题，内容包括为什么使用主题，如何定义主题及其外观，在主题中定义外观和样式表的操作，以及利用主题实现换肤功能；最后介绍了母版页的使用，内容包括母版页和内容页的概念、创建母版页和内容页的方法、从内容页访问母版页成员的方法等。通过本章的学习，读者应该能够掌握页面布局及修饰的方法，以便能够更好地美化页面。

7.6　思考和练习

1. 简述 CSS 的概念。
2. 简述 CSS 样式在 ASP.NET 中的引用方式。
3. 简述盒子模型的概念。
4. 为什么要使用主题？如何利用主题实现换肤功能？
5. 简述母版页和内容页的概念，以及创建方法。

第8章　ADO.NET数据访问技术

ASP.NET 应用程序的数据访问是通过 ADO.NET 进行的，ADO.NET 可以使 Web 应用程序从各种数据源中快速访问数据。从传统的数据库到 XML 数据存储，各种各样的数据源都能连接到 ADO.NET，从而更加灵活地访问数据，减少访问数据所需的代码，提高了开发效率和 Web 应用程序的性能。

本章首先介绍 ADO.NET 的基本知识，然后再详细介绍在 ASP.NET 中的几种数据访问方法，而有关数据绑定的内容则第 9 章再进行介绍。

本章的学习目标：

- 了解 ADO.NET 的基本知识。
- 掌握 ADO.NET 与数据库的连接方法。
- 掌握利用 Command 访问数据库的方法。
- 掌握利用 DataAdapter 访问数据库的方法。

8.1　ADO.NET 概述

ADO.NET 是.NET Framework 提供的数据访问的类库，ADO.NET 对 Microsoft SQL Server、Oracle 和 XML 等数据源提供一致的访问。应用程序可以使用 ADO.NET 连接到这些数据源，并检索和更新所包含的数据。

ADO.NET 用于数据访问的类库包含.NET Framework 数据提供程序和 DataSet 两个组件。.NET Framework 数据提供程序与 DataSet 之间的关系如图 8-1 所示。

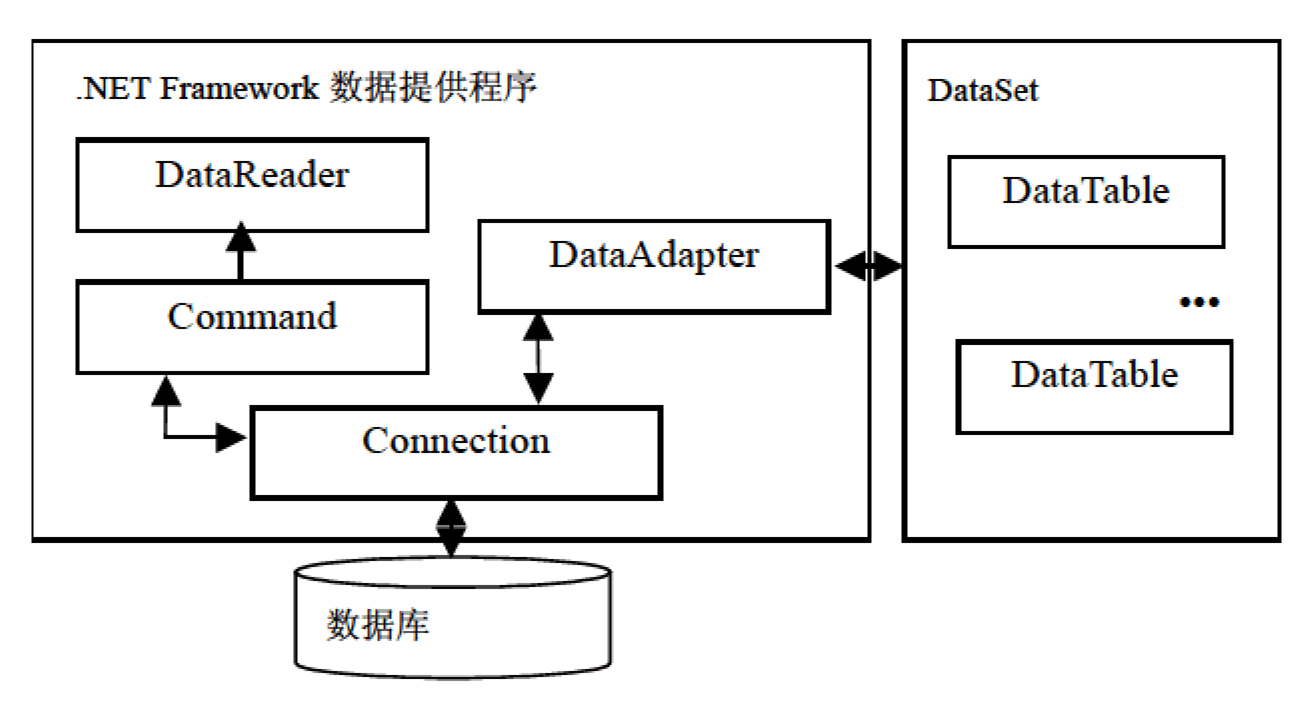


图 8-1　ADO.NET 的组成

.NET Framework 数据提供程序包含以下 4 个核心类。

- Connection：建立与数据源的连接。

- Command：对数据源执行操作命令，用于修改数据、查询数据和运行存储过程等。
- DataReader：从数据源获取返回的数据。
- DataAdapter：用数据源数据填充 DataSet，并可以处理数据更新。

DataSet 是 ADO.NET 的断开式结构的核心组件。设计 DataSet 的目的是为了实现独立于任何数据源的数据访问，可以把它看成是内存中的数据库，是专门用来处理数据源中读出的数据。

DataSet 的优点就是离线式，一旦读取到数据库中的数据后，就在内存中建立数据库的副本。在此之后的操作，直到执行更新命令为止，所有的操作都是在内存中完成的。不管底层的数据库是哪种形式，DataSet 的行为都是一致的。

DataSet 是数据表(DataTable)的集合，它可以包含任意多个数据表，而且每个 DataSet 中的数据表对应一个数据源中的数据表(Table)或者是数据视图(View)。DataSet 的结构如图 8-2 所示。

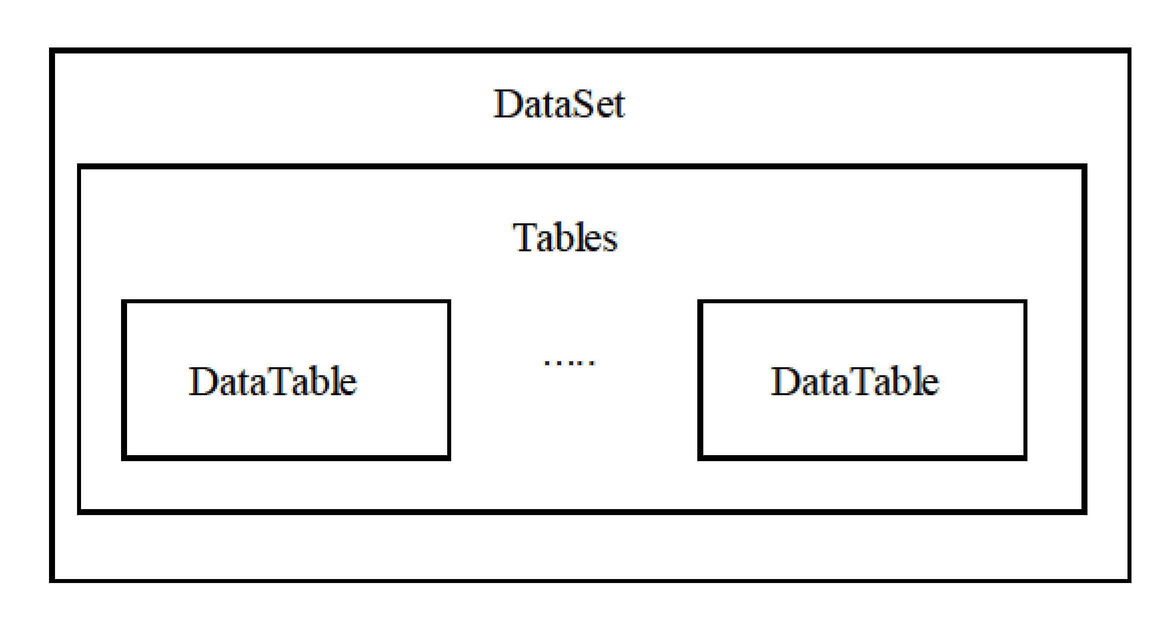


图 8-2　DataSet 的结构

ASP.NET 数据访问程序的开发流程有以下几个步骤。

(1) 利用 Connection 对象创建数据连接。

(2) 利用 Command 对象数据源执行 SQL 命令。

(3) 利用 DataReader 对象读取数据源的数据。

(4) DataSet 对象与 DataAdapter 对象配合，完成数据的查询和更新操作。

在 ADO.NET 中，连接数据源有 4 种接口：SQLClient、OracleClient 、ODBC、OLEDB。其中 SQLClient 是 Microsoft SQL Server 数据库专用连接接口，OracleClient 是 Oracle 数据库专用的连接接口，ODBC 和 OLEDB 可用于其他数据源的连接。在应用程序中使用任何一种连接接口时，必须在后台代码中引用对应的名称空间，类的名称也随之发生变化，如表 8-1 所示。

表 8-1　数据连接方式名称空间与对应的类名称

名 称 空 间	对应的类名称
System.Data.SqlClient	SqlConnection；SqlCommand；SqlDataReader；SqlDataAdapter
System.Data.Odbc	OdbcConnection；OdbcCommand；OdbcDataReader；OdbcDataAdapter
System.Data.OleDb	OleDbConnection；OleDbCommand；OleDbDataReader；OleDbDataAdapter
System.Data.OracleClient	OracleConnection；OracleCommand；OracleDataReader；OracleDataAdapter

8.2　使用 Connection 连接数据库

本书数据源以 Microsoft SQL Server 2012 数据库为例。也就是说，使用 SQLClient 连接接口，访问数据库使用 SqlConnection、SqlCommand、SqlDataReader 和 SqlDataAdapter 对象。

在建立数据库连接之前，首先在 Web.Config 配置文件中建立一个连接字符串，然后建立数据库连接。

【例 8-1】演示如何建立 Microsoft SQL Server 2012 数据库连接。

(1) 运行 VS，新建一个名为 WebSite8 的 ASP.NET 网站。

(2) 在【解决方案资源管理器】中，右击网站名，选择【添加】|【新建项】命令，在弹出的对话框中选择【数据】|【SQL Server 数据库】模板，更改名称为 MyDatabase.mdf，创建数据库，如图 8-3 所示。

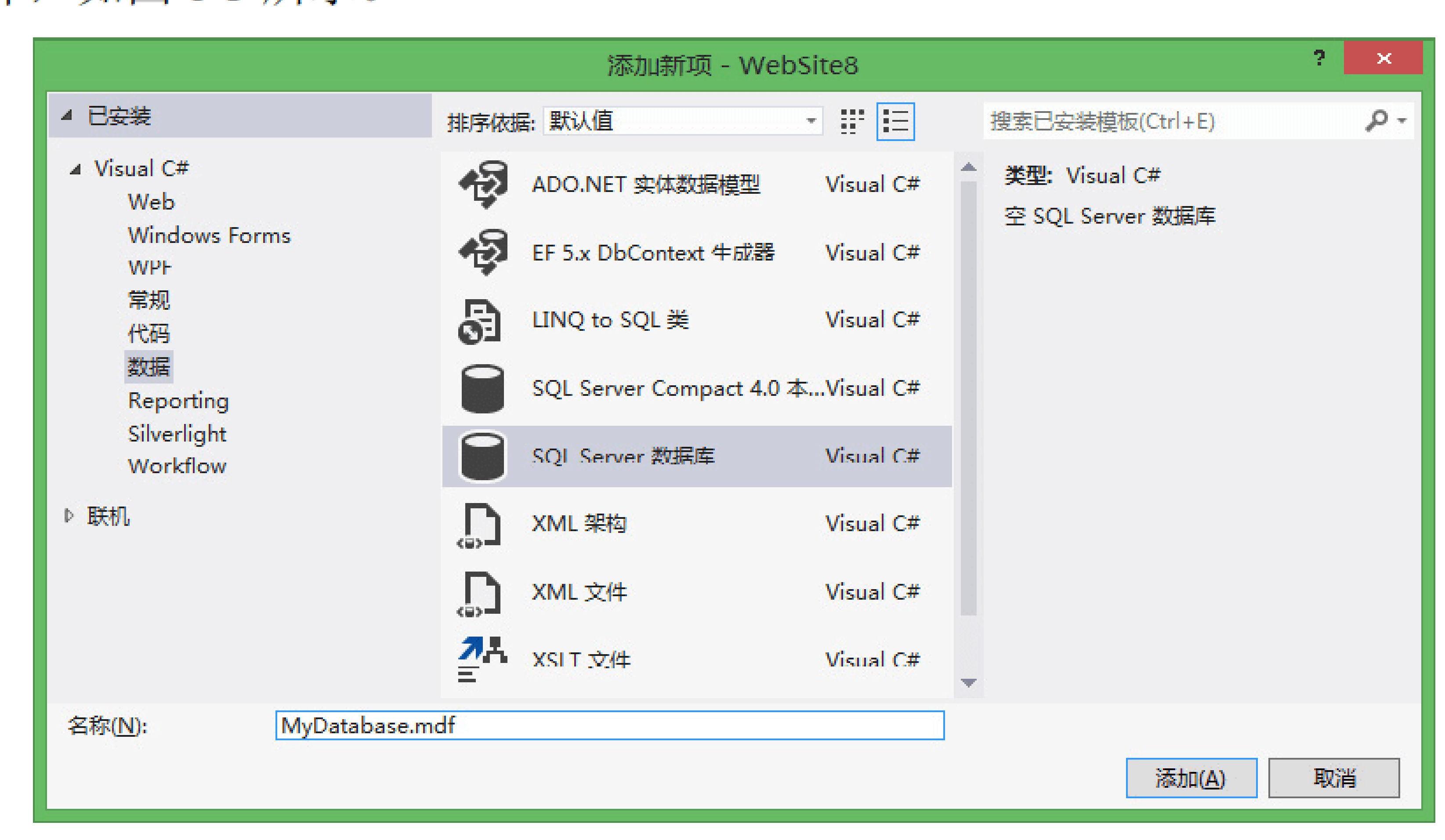


图 8-3　新建数据库

(3) 单击【添加】按钮，弹出如图 8-4 所示的对话框，单击【确定】按钮，将数据库 MyDatabase.mdf 保存到 App_Data 文件夹中。

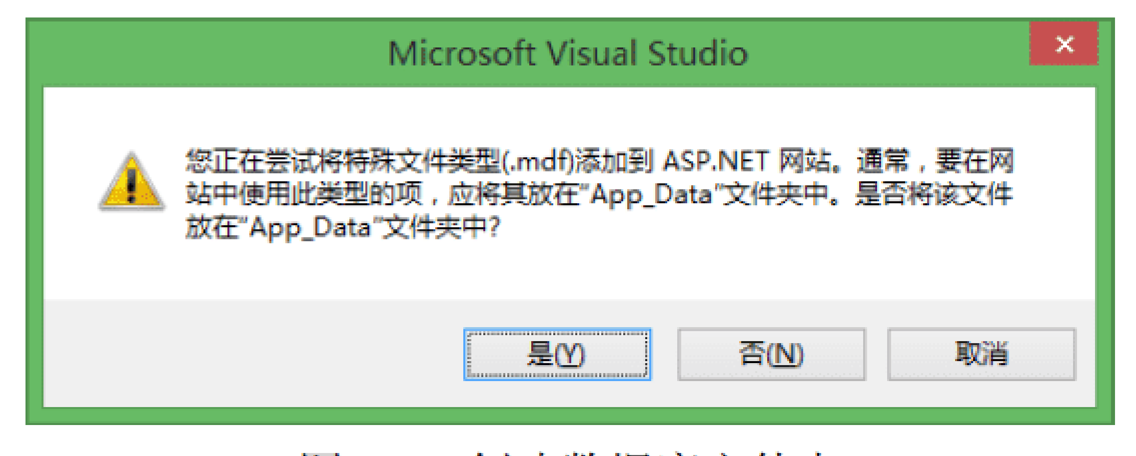


图 8-4　创建数据库文件夹

(4) 在【数据库资源管理器】中，双击数据库名 MyDatabase.mdf，数据库资源管理器显示如图 8-5 所示。

(5) 右击【表】结点，创建 student 表，表的结构和记录如表 8-2 所示。

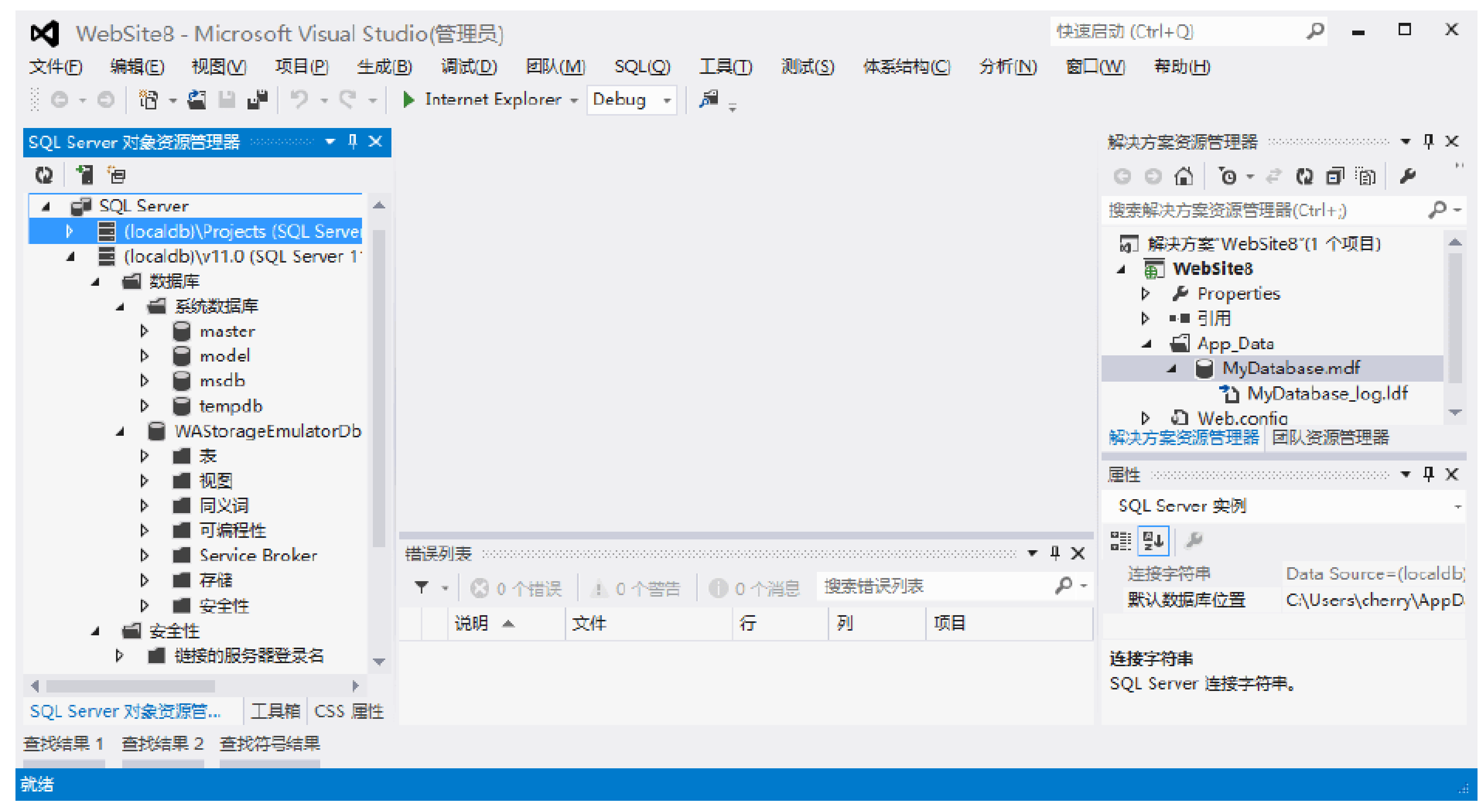


图 8-5　数据库资源管理器

表 8-2　student 表的结构和记录

No (varchar，10) (编号，主键)	Name (varchar，50) (姓名)	Sex (char，2) (性别)	birth (datetime) (出生日期)	Address (varchar，50) (地址)	Photo (varchar，50) (照片)
1	张兵	男	1989-6-29	北京	1.jpg
2	李明	男	1988-1-2	上海	2.jpg
3	王彬	女	1989-9-25	信阳	3.jpg
4	陈小菲	女	1988-6-8	苏州	4.jpg
5	王小燕	女	1989-12-11	辽阳	5.jpg
6	陈芬芳	女	1985-7-21	广州	6.jpg
7	刘华南	男	1990-3-15	广州	7.jpg
8	翟力	女	1989-6-11	深圳	8.jpg
9	丽丽	女	1989-6-1	沈阳	9.jpg

(6) 打开 web.config 配置文件，将<connectionStrings/>标记用下面代码替换。

```
<connectionStrings>
<add name="ConnectionString" connectionString="Data Source=.\SQLEXPRESS;
AttachDbFilename=|DataDirectory|\mydatabase.mdf;Integrated Security=True;User Instance=True"/>
</connectionStrings>
```

其中，Data Source 表示 SQL Server 数据库服务器名称，AttachDbFilename 表示数据库的路径和文件名，|DataDirectory| 表示网站默认数据库路径 App_Data。

(7) 在网站中添加一个名为 connection.aspx 的网页，切换到【设计】视图，向该页面拖放一个 Label 控件，使用默认控件名称，然后在 connection.aspx.cs 页面中添加下面代码。

```
//引用数据库访问名称空间
using System.Data.SqlClient;
```

```
using System.Configuration;
…
protected void Page_Load(object sender, EventArgs e)
{
  //从 web.config 配置文件取出数据库连接串
  string sqlconnstr =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;
  //建立数据库连接对象
  SqlConnection sqlconn = new SqlConnection(sqlconnstr);
  //打开连接对象
  sqlconn.Open();
  Label1.Text = "成功建立 SQL Server 2012 数据库连接";
  //关闭连接对象
  sqlconn.Close();
  sqlconn = null;
}
```

(8) 程序运行效果如图 8-6 所示。

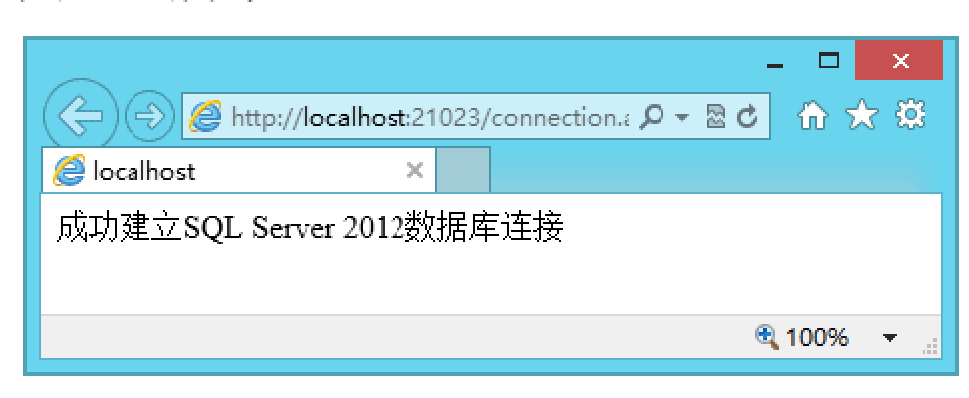


图 8-6　connection.aspx 运行效果

在访问数据库的数据之前，需要使用 Connection 对象的 Open 方法把数据库打开，并在完成数据库的操作之后使用 Connection 对象的 Close 方法将数据库关闭。

8.3　使用 Command 对象执行数据库命令

Command 对象是用来执行数据库操作命令的，比如对数据库中数据表记录的查询、增加、修改或删除等都是要通过 Command 对象来实现的。一个数据库操作命令可以用 SQL 语句来表达，包括 SELECT 语句、UPDATE 语句、DELETE 语句、INSERT 语句等。Command 对象可以传递参数并返回值，同时 Command 也可以调用数据库中的存储过程。

像 Connection 对象一样，对于操作 SQL Server 数据库而言，使用 SqlCommand 对象。Command 对象的属性如表 8-3 所示。

表 8-3　Command 对象的属性

属　性	说　明
Connection	包含了数据库连接信息的 Connection 对象
CommandText	要运行的 SQL 命令
CommandType	命令类型
Parameters	Parameters 对象集合

8.3.1　使用 Command 对象从数据库中查询数据

使用 Command 对象查询数据库数据的一般步骤为：先建立数据库连接；然后创建 Command 对象，并设置它的 Connection 和 CommandText 两个属性，分别表示数据库连接和需要执行的 SQL 命令；接下来使用 Command 对象的 ExecuteReader 方法，把返回结果放在 DataReader 对象中；最后通过循环，显示数据库查询结果。

【例 8-2】在【例 8-1】基础上，介绍如何使用 Command 对象查询数据库的数据。

(1) 新建 WebSite8 网站，然后在其中添加一个名为 command_query.aspx 的网页，切换到【设计】视图，向该页面拖放一个 Label 控件，使用默认控件名称。

(2) 在 command_query.aspx.cs 页面中添加下面代码。

```
//引用数据库访问名称空间
using System.Data.SqlClient;
using System.Configuration;
• • •
protected void Page_Load(object sender, EventArgs e)
        {
            string sqlconnstr = 
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;
            SqlConnection sqlconn = new SqlConnection(sqlconnstr);
            //建立 Command 对象
            SqlCommand sqlcommand = new SqlCommand();
            //给 sqlcommand 的 Connection 属性赋值
            sqlcommand.Connection = sqlconn;
            //打开连接
            sqlconn.Open();
            //SQL 命令赋值
            sqlcommand.CommandText = "select * from student";
            //建立 DataReader 对象，并返回查询结果
            SqlDataReader sqldatareader = sqlcommand.ExecuteReader();
            //逐行遍历查询结果
            while (sqldatareader.Read())
            {
                Label1.Text += sqldatareader.GetString(0) + "  ";
                Label1.Text += sqldatareader.GetString(1) + "  ";
                Label1.Text += sqldatareader.GetString(2) + "  ";
                Label1.Text += sqldatareader.GetDateTime(3) + "  ";
                Label1.Text += sqldatareader.GetString(4) + "  ";
                Label1.Text += sqldatareader.GetString(5) + "  <br>";
            };
            sqlcommand = null;
            sqlconn.Close();
            sqlconn = null;
        }
```

(3) 程序运行效果如图 8-7 所示。

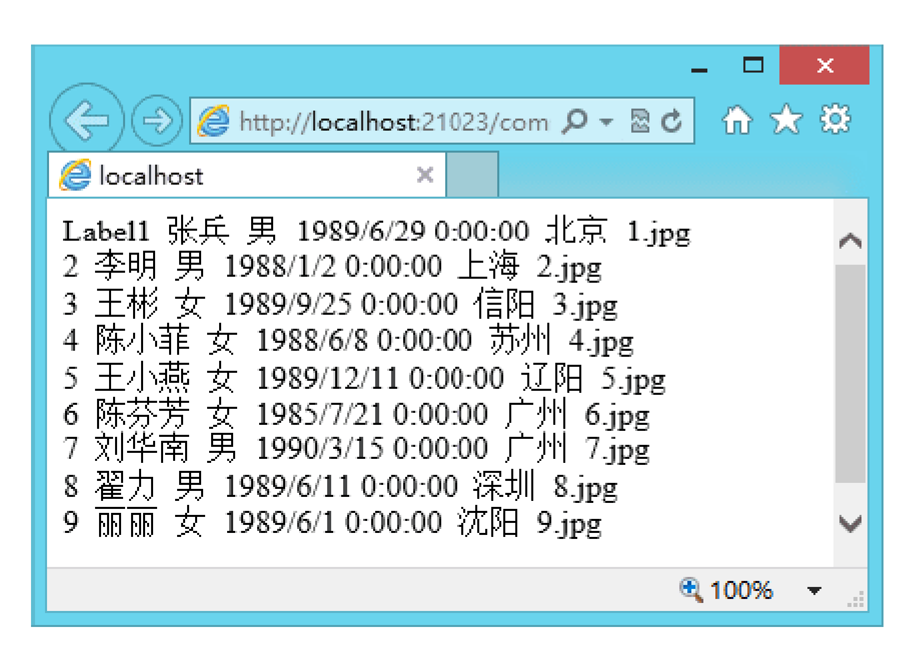


图 8-7　command_query.aspx 运行效果

8.3.2　使用 Command 对象向数据库插入数据

使用 Command 对象增加数据库数据的一般步骤为：先建立数据库连接；然后创建 Command 对象，设置它的 Connection 和 CommandText 两个属性，并使用 Command 对象的 Parameters 属性来设置输入参数；最后使用 Command 对象的 ExecuteNonquery 方法执行数据库数据增加命令，其中 ExecuteNonquery 方法表示要执行的是没有返回数据的命令。

【例 8-3】演示如何使用 Command 对象向数据库中插入数据记录。

(1) 在【解决方案资源管理器】中，右击网站名 WebSite8，选择【新建文件夹】命令，新建文件夹，改名为 images，用于存放学生照片。

(2) 在 WebSite8 网站中添加一个名为 command_insert.aspx 的网页。

(3) 将 command_insert 页面设计为如图 8-8 所示的样子。



图 8-8　command_insert.aspx 页面的设计效果

对应【源】视图中的代码如下。

```
<table style="width: 320px; height: 240px">
        <tr>
        <td style="width: 100px; text-align: right"> 学号：</td>
        <td style="width: 220px">
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox></td>    </tr>
        <tr>
        <td style="width: 100px; text-align: right"> 姓名：</td>
        <td style="width: 220px">
 <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox></td>    </tr>
        <tr>
        <td style="width: 100px; text-align: right"> 性别：</td>
        <td style="width: 220px">
          <asp:DropDownList ID="DropDownList1" runat="server">
                <asp:ListItem Selected="True">男</asp:ListItem>
                <asp:ListItem>女</asp:ListItem>
          </asp:DropDownList>    </td>    </tr>
        <tr>
        <td style="width: 100px; text-align: right">   出生日期：</td>
```

```
        <td style="width: 220px">
            <asp:TextBox ID="TextBox3" runat="server"></asp:TextBox></td> </tr>
        <tr>
        <td style="width: 100px; text-align: right"> 地址：</td>
        <td style="width: 220px">
            <asp:TextBox ID="TextBox4" runat="server"></asp:TextBox></td>   </tr>
        <tr>
        <td style="width: 100px; text-align: right"> 照片：</td>
        <td style="width: 220px">
            <asp:FileUpload ID="FileUpload1" runat="server" /></td> </tr>
        <tr>
        <td colspan="2" style="text-align: center">
      <asp:Button ID="Button1" runat="server" Text="提交" OnClick="Button1_Click" /></td> </tr>
    </table>
    <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
```

(4) 双击设计视图中的【提交】按钮，添加如下所示的后台代码。

```
    protected void Button1_Click(object sender, EventArgs e)
    {
    string sqlconnstr = ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;
      SqlConnection sqlconn = new SqlConnection(sqlconnstr);
      //建立 Command 对象
      SqlCommand sqlcommand = new SqlCommand();
      sqlcommand.Connection = sqlconn;
      //把 SQL 语句赋给 Command 对象
      sqlcommand.CommandText = "insert into student(No,Name,Sex,birthday,Adress,Photo)values
(@No,@Name,@Sex,@birthday,@Adress,@Photo)";
      sqlcommand.Parameters.AddWithValue("@No",TextBox1.Text);
      sqlcommand.Parameters.AddWithValue("@Name",TextBox2.Text);
      sqlcommand.Parameters.AddWithValue("@Sex",DropDownList1.Text);
      sqlcommand.Parameters.AddWithValue("@birthday",TextBox3.Text);
      sqlcommand.Parameters.AddWithValue("@Adress",TextBox4.Text);
      sqlcommand.Parameters.AddWithValue("@Photo",FileUpload1.FileName);
      try
      {
           //打开连接
           sqlconn.Open();
           //执行 SQL 命令
           sqlcommand.ExecuteNonQuery();
           //把学生的照片上传到网站的 images 文件夹中
         if (FileUpload1.HasFile == true)
           {
               FileUpload1.SaveAs(Server.MapPath(("~/image/") + FileUpload1.FileName));
           }
           Label1.Text = "成功增加记录";
      }
      catch (Exception ex)
      {
           Label1.Text = "错误原因："+ ex.Message;
      }
      finally
```

```
        {
            sqlcommand = null;
            sqlconn.Close();
            sqlconn = null;
        }
    }
```

(5) 程序运行效果如图 8-9 所示。

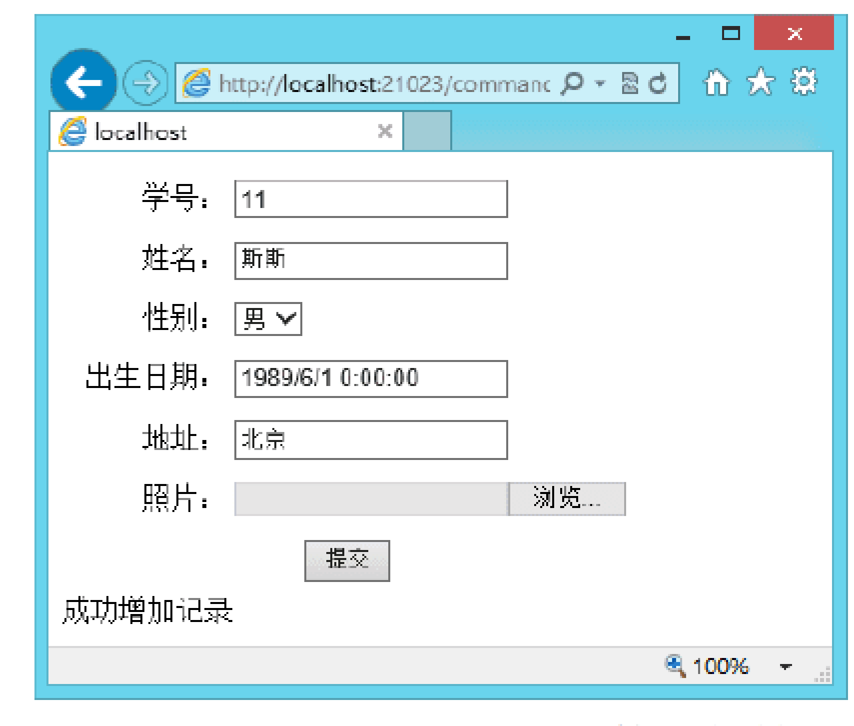


图 8-9　command_insert.aspx 的运行效果

8.3.3　使用 Command 对象删除数据库中的数据

使用 Command 对象删除数据库数据的一般步骤为：先建立数据库连接；然后创建 Command 对象，设置它的 Connection 和 CommandText 两个属性，并使用 Command 对象的 Parameters 属性来传递参数；最后使用 Command 对象的 ExecuteNonquery 方法执行数据库数据删除命令。

【例 8-4】使用 Command 对象删除数据。

(1) 在 WebSite8 网站中添加一个名为 command_delete.aspx 的网页。

(2) 向 command_delete 页面添加 2 个 Label 控件，1 个 TextBox 控件，1 个 Button 控件，其中 Button 控件作为【删除】按钮。

(3) 双击设计视图中的【删除】按钮，添加如下所示的后台代码。

```
    using System.Data.SqlClient;
    using System.Configuration;

    • • •
    protected void Button1_Click(object sender, EventArgs e)
    {
        int intDeleteCount;
        string sqlconnstr =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;
        SqlConnection sqlconn = new SqlConnection(sqlconnstr);
        //建立 Command 对象
        SqlCommand sqlcommand = new SqlCommand();
        //给 Command 对象的 Connection 和 CommandText 属性赋值
        sqlcommand.Connection = sqlconn;
        sqlcommand.CommandText = "delete from student where no=@no";
        sqlcommand.Parameters.AddWithValue("@no",TextBox1.Text);
        try
        {
            sqlconn.Open();
            intDeleteCount=sqlcommand.ExecuteNonQuery();
            if (intDeleteCount>0)
                Label1.Text = "成功删除记录";
            else
                Label1.Text = "该记录不存在";
        }
        catch (Exception ex)
```

```
        {
          Label1.Text = "错误原因："+ex.Message;
        }
        finally
        {
            sqlcommand = null;
            sqlconn.Close();
            sqlconn = null;
        }
    }
```

(4) 程序运行效果如图 8-10 所示。

图 8-10　command_delete.aspx 的运行效果

8.3.4　使用 Command 对象修改数据库的数据

使用 Command 对象修改数据库数据的一般步骤为：先建立数据库连接；然后创建 Command 对象，设置它的 Connection 和 CommandText 两个属性，并使用 Command 对象的 Parameters 属性来传递参数；接下来使用 Command 对象的 ExecuteNonquery 方法执行数据库数据修改命令。【例 8-5】同时说明存储过程的调用方法。

【例 8-5】演示如何使用 Command 对象修改数据。

(1) 打开 WebSite8 网站，为 MyDatabase.mdf 数据库建立名为 update_student 的存储过程。在【数据库资源管理器】中，右击【存储过程】结点，如图 8-11 所示。

选择【添加新存储过程】命令，在存储过程定义窗口中，添加如下代码，最后单击工具栏中的【保存】按钮，保存存储过程到数据库中。

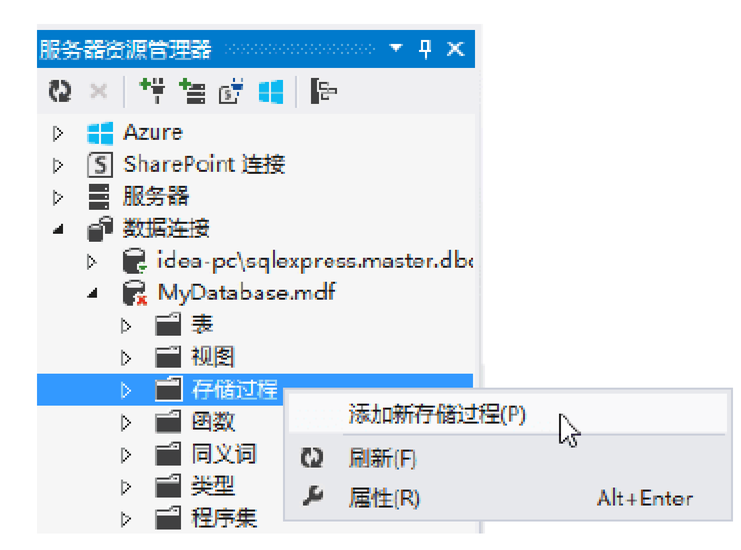


图 8-11　数据库资源管理器

```
    CREATE PROCEDURE [dbo].[update_student]
    //入口参数
     @NO nvarchar(10),
     @Name nvarchar(50),
     @Sex    nvarchar(10),
     @birthday datetime,
     @Adress nvarchar(50),
     @Photo nvarchar(50)
    AS
    //修改学生记录
     update student set
No=@No,Name=@Name,Sex=@Sex,birthday=@birthday,Adress=@Adress,Photo=@Photo where
No=@No
     RETURN 0
```

(2) 在 WebSite8 网站中添加一个名为 command_update.aspx 的网页，将 command_update 页面设计为如图 8-8 所示的形式。

(3) 双击设计视图中的【提交】按钮，添加如下所示的后台代码。

```
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
• • •
protected void Button1_Click(object sender, EventArgs e)
        {
            int intUpdateCount;
            string sqlconnstr =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;
            SqlConnection sqlconn = new SqlConnection(sqlconnstr);
            //建立 Command 对象
            SqlCommand sqlcommand = new SqlCommand();
            sqlcommand.Connection = sqlconn;
            //把存储过程名称赋给 Command 对象的 CommandText 属性
            sqlcommand.CommandText = "update_student";
            //说明命令类型为存储过程
            sqlcommand.CommandType = CommandType.StoredProcedure;
            sqlcommand.Parameters.AddWithValue("@No", TextBox1.Text);
            sqlcommand.Parameters.AddWithValue("@Name", TextBox2.Text);
            sqlcommand.Parameters.AddWithValue("@Sex", DropDownList1.Text);
            sqlcommand.Parameters.AddWithValue("@birthday", TextBox3.Text);
            sqlcommand.Parameters.AddWithValue("@Adress", TextBox4.Text);
            sqlcommand.Parameters.AddWithValue("@Photo", FileUpload1.FileName);
            try
            {
                //打开连接
                sqlconn.Open();
                //执行 SQL 命令
                intUpdateCount = sqlcommand.ExecuteNonQuery();
                //把学生的照片上传到网站的 images 文件夹中
                if (FileUpload1.HasFile == true)
                {
                    FileUpload1.SaveAs(Server.MapPath(("~/images/") +
FileUpload1.FileName));
                }
                if (intUpdateCount > 0)
                    Label1.Text = "成功修改记录";
                else
                    Label1.Text = "该记录不存在";
            }
            catch (Exception ex)
            {
                Label1.Text = "错误原因：" + ex.Message;
            }
            finally
            {
                sqlcommand = null;
                sqlconn.Close();
                sqlconn = null;
            }
        }
```

(4) 程序运行效果如图 8-12 所示。

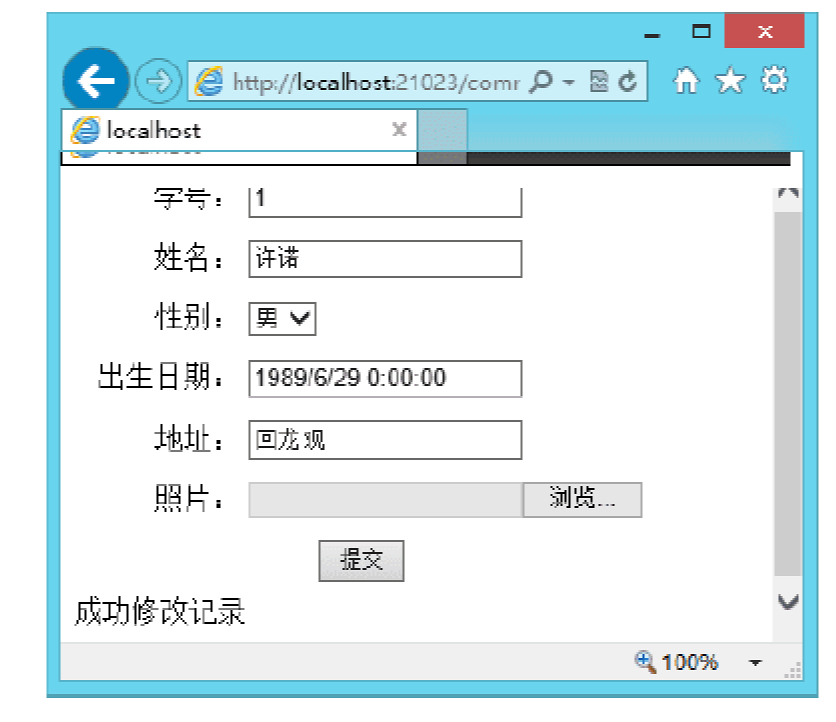


图 8-12　command_ update.aspx 的运行效果

8.3.5　数据库事务处理

对于数据库管理系统来说，如果没有显式定义事务的开始和结束，就默认一条 SQL 语句为一个单独事务，多数情况下采用这种默认方式就足够了。但是，有时需要将一组 SQL 语句作为一个事务，要不全做，要不全不做。

在 ASP.NET 中，可以使用 Connection 和 Transaction 对象开始、提交和回滚事务。一般步骤为：调用 Connection 对象的 BeginTransaction 方法来标记事务的开始，BeginTransaction 方法返回对 Transaction 的引用；将 Transaction 对象赋给 Command 的 Transaction 属性；执行事务操作；如果事务操作成功，使用 Transaction 对象的 Commit 方法提交事务，否则，使用 Rollback 方法回滚事务。

【例 8-6】演示事务处理。

(1) 在 WebSite8 网站中添加一个名为 transaction.aspx 的网页。

(2) 向 transaction 页面添加 1 个 Label 控件，1 个 Button 控件，其中 Button 控件作为【事务提交】按钮。

(3) 双击设计视图中的【事务提交】按钮，添加如下所示的后台代码。

```
    using System.Data.SqlClient;
    • • •
    protected void Button1_Click(object sender, EventArgs e)
    {
        string sqlconnstr =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;
        SqlConnection sqlconn = new SqlConnection(sqlconnstr);
        sqlconn.Open();
        //开始事务
        SqlTransaction tran=sqlconn.BeginTransaction();
        SqlCommand sqlcommand = new SqlCommand();
        sqlcommand.Connection = sqlconn;
        sqlcommand.Transaction = tran;
        try
        {
            sqlcommand.CommandText = "update student set Adress='beijing' where No=1";
            sqlcommand.ExecuteNonQuery();
            sqlcommand.CommandText = "update student set Adress='zhengzhou' where No=2";
            sqlcommand.ExecuteNonQuery();
            tran.Commit();
            Label1.Text = "事务提交成功";
        }
        catch (Exception ex)
        {
```

```
                tran.Rollback();
                Label1.Text = "事务提交失败：" + ex.Message;
            }
            finally
            {
                sqlcommand = null;
                sqlconn.Close();
                sqlconn = null;
            }
        }
```

从 8.3 节例子中可以看出，Command 对象的 ExecuteNonQuery 方法在执行数据更新 SQL 语句(如 INSERT，UPDATE 或 DELETE)时使用，这些语句的共同特点是没有返回数据。此外，ExecuteNonQuery 方法可以返回一个整数，表示已经执行语句在数据库中影响数据的行数。

如果需要执行有返回结果的 SQL 语句(如 SELECT)，那么就需要使用 Command 对象的 ExecuteReader 方法，并将执行结果放到 DataReader 中。这个对象是一个专门读取数据的对象，除了能做读取数据工作之外，其他什么也不能做，所以这是一种简单的读取数据的方法。

8.4　使用 DataAdapter 对象执行数据库命令

对于 SQL SERVER 接口，使用的是 SqlDataAdapter 对象，在使用 DataAdapter 对象时，只需分别设置表示 SQL 命令和数据库连接的两个参数，就可以通过它的 Fill 方法把查询结果放在一个 DataSet 对象中。

在一个 DataSet 对象实例中，可以包含多个 DataTable，而一个 DataTable 可以包含多个 DataRow。

当把一个 DataSet 中的一个数据表复制到一个 DataTable 中之后，可以通过对 DataTable 数据的访问来实现对 DataSet 中数据的访问。除此之外，还可以通过修改 DataTable 中的数据来更新 DataSet。

DataRow 表示 DataTable 的数据行，一个 DataTable 中的数据行会有很多。针对一个 DataTable，它的 Rows 属性表示这个表的所有数据行，是一个集合，类名为 DataRowCollection，它的每个元素的类型是 DataRow。

通过 8.4 节的例子说明使用 DataAdapter 和 DataSet 读取和修改数据库数据的方法。

8.4.1　使用 DataAdapter 对象查询数据库的数据

使用 DataAdapter 对象查询数据库数据的一般步骤为：首先建立数据库连接；然后利用数据库连接和 SELECT 语句建立 DataAdapter 对象，并使用 DataAdapter 对象的 Fill 方法把查询结果放在 DataSet 对象的一个数据表中；接下来将该数据表复制到 DataTable 对象

中；最后实现对 DataTable 对象中数据的查询。

【例 8-7】演示如何使用 DataAdapter 对象查询数据库的数据。

(1) 在 WebSite8 网站中添加一个名为 DataAdapter_select.aspx 的网页，切换到【设计】视图，向该页面拖放一个 Label 控件，使用默认控件名称。

(2) 在 DataAdapter _select.aspx.cs 页面中添加如下代码。

```
    //引用数据库访问相关名称空间
    using System.Configuration;
    using System.Data;
    using System.Data.SqlClient;
    • • •
    protected void Page_Load(object sender, EventArgs e)
    {
        string sqlconnstr =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;
        SqlConnection sqlconn = new SqlConnection(sqlconnstr);
        //建立 DataSet 对象
        DataSet ds = new DataSet();
        //建立 DataTable 对象
        DataTable dtable;
        //建立 DataRowCollection 对象
        DataRowCollection coldrow;
        //建立 DataRow 对象
        DataRow drow;
        //打开连接
        sqlconn.Open();
        //建立 DataAdapter 对象
        SqlDataAdapter sqld = new SqlDataAdapter("select * from student", sqlconn);
        //用 Fill 方法返回的数据，填充 DataSet，数据表取名为 tabstudent
        sqld.Fill(ds, "tabstudent");
        //将数据表 tabstudent 的数据复制到 DataTable 对象
        dtable = ds.Tables["tabstudent"];
        //用 DataRowCollection 对象获取这个数据表的所有数据行
        coldrow = dtable.Rows;
        //逐行遍历，取出各行的数据
        for (int inti = 0; inti < coldrow.Count; inti++)
        {
            drow = coldrow[inti];
            Label1.Text += "学号：" + drow[0];
            Label1.Text += " 姓名：" + drow[1];
            Label1.Text += " 性别：" + drow[2];
            Label1.Text += " 出生日期：" + drow[3];
            Label1.Text += " 地址：" + drow[4] + "<br />";
        }
        sqlconn.Close();
        sqlconn = null;
    }
```

(3) 程序运行效果如图 8-13 所示。

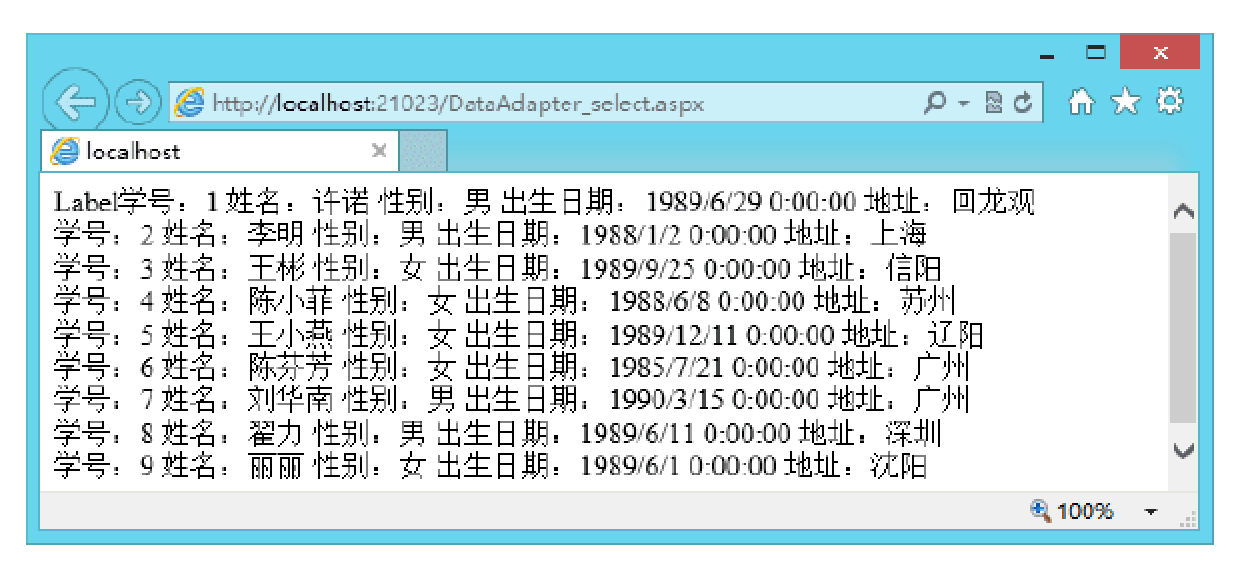


图 8-13　DataAdapter _select.aspx 运行效果

关于显示 DataSet 中的数据还有更简单的方法，就是绑定 GridView 控件等，详细内容在第 9 章中进行介绍。

8.4.2　使用 DataAdapter 对象修改数据库的数据

使用 DataAdapter 对象修改数据库数据的一般步骤为：首先建立数据库连接；然后利用数据库连接和 SELECT 语句建立 DataAdapter 对象；并配置它的 UpdateCommand 属性，定义修改数据库的 UPDATE 语句；使用 DataAdapter 对象的 Fill 方法把 SELECT 语句的查询结果放在 DataSet 对象的数据表中；接下来将该数据表复制到 DataTable 对象中；最后实现对 DataTable 对象中数据的修改，并通过 DataAdapter 对象的 Update 方法向数据库提交修改数据。

【例 8-8】演示如何使用 DataAdapter 对象修改数据库的数据。

(1) 在 WebSite8 网站中添加一个名为 DataAdapter_update.aspx 的网页。

(2) 向 DataAdapter_ update 页面添加 1 个 Label 控件，1 个 Button 控件，其中 Button 控件作为【更新】提交按钮。

(3) 双击设计视图中的【更新】按钮，添加如下所示的后台代码。

```
    using System.Data.SqlClient;
    • • •
    protected void Button1_Click(object sender, EventArgs e)
    {
        string sqlconnstr =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;
        SqlConnection sqlconn = new SqlConnection(sqlconnstr);
        //建立 DataSet 对象
        DataSet ds = new DataSet();
        //建立 DataTable 对象
        DataTable dtable;
        //建立 DataRowCollection 对象
        DataRowCollection coldrow;
        //建立 DataRow 对象
        DataRow drow;
        //打开连接
        sqlconn.Open();
        //建立 DataAdapter 对象
        SqlDataAdapter sqld = new SqlDataAdapter("select * from student", sqlconn);
        //自己定义 Update 命令，其中@NAME，@NO 是两个参数
```

```
        sqld.UpdateCommand = new SqlCommand("UPDATE student SET NAME = @NAME
    WHERE NO = @NO", sqlconn);
        //定义@NAME 参数，对应于 student 表的 NAME 列
        sqld.UpdateCommand.Parameters.Add("@NAME", SqlDbType.VarChar, 50, "NAME");
        //定义@NO 参数，对应于 student 表的 NO 列，而且@NO 是修改前的原值
        SqlParameter parameter = sqld.UpdateCommand.Parameters.Add("@NO", SqlDbType.VarChar,
10);
        parameter.SourceColumn = "NO";
        parameter.SourceVersion = DataRowVersion.Original;
        //用 Fill 方法返回的数据，填充 DataSet，数据表取名为 tabstudent
        sqld.Fill(ds, "tabstudent");
        //将数据表 tabstudent 的数据复制到 DataTable 对象
        dtable = ds.Tables["tabstudent"];
        //用 DataRowCollection 对象获取这个数据表的所有数据行
        coldrow = dtable.Rows;
        //修改操作，逐行遍历，取出各行的数据
        for (int inti = 0; inti < coldrow.Count; inti++)
        {
            drow = coldrow[inti];
            //给每位学生姓名后加上字母 A
            drow[1]=drow[1]+"A";
        }
        //提交更新
        sqld.Update(ds, "tabstudent");
       sqlconn.Close();
        sqlconn = null;
        Label1.Text = "更新成功";
    }
```

(4) 程序运行效果如图 8-14 所示。

图 8-14　DataAdapter_ update.aspx 的运行效果

8.4.3　使用 DataAdapter 对象增加数据库的数据

使用 DataAdapter 对象增加数据库数据的一般步骤为：首先建立数据库连接；然后利用数据库连接和 SELECT 语句建立 DataAdapter 对象；并建立 CommandBuilder 对象自动生成 DataAdapter 的 Command 命令，否则就要自己给 UpdateCommand、InsertCommand、DeleteCommand 属性定义 SQL 更新语句；使用 DataAdapter 对象的 Fill 方法把 SELECT 语句的查询结果放在 DataSet 对象的数据表中；接下来将该数据表复制到 DataTable 对象中；最后实现对 DataTable 对象中数据的增加，并通过 DataAdapter 对象的 Update 方法向数据库提交数据。

【例 8-9】演示如何使用 DataAdapter 对象增加一条学生记录。

(1) 在 WebSite8 网站中添加一个名为 DataAdapter_insert.aspx 的网页。

(2) 向 DataAdapter_insert 页面添加 1 个 Label 控件，1 个 Button 控件，其中 Button 控件作为【增加】提交按钮。

(3) 双击设计视图中的【增加】按钮，添加如下后台代码。

```
    using System.Configuration;
```

```
    using System.Data;
    using System.Data.SqlClient;
    • • •
    protected void Button1_Click(object sender, EventArgs e)
    {
      string sqlconnstr =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;
      SqlConnection sqlconn = new SqlConnection(sqlconnstr);
      //建立 DataSet 对象
      DataSet ds = new DataSet();
      //建立 DataTable 对象
      DataTable dtable;
      //建立 DataRow 对象
      DataRow drow;
      //打开连接
      sqlconn.Open();
      //建立 DataAdapter 对象
      SqlDataAdapter sqld = new SqlDataAdapter("select * from student", sqlconn);
      //建立 CommandBuilder 对象来自动生成 DataAdapter 的 Command 命令，否则就要自己编
写
      //Insertcommand ,deletecommand , updatecommand 命令。
      SqlCommandBuilder cb = new SqlCommandBuilder(sqld);
    //用 Fill 方法返回的数据，填充 DataSet，数据表取名为 tabstudent
      sqld.Fill(ds, "tabstudent");
      //将数据表 tabstudent 的数据复制到 DataTable 对象
      dtable = ds.Tables["tabstudent"];
      //增加新记录
      drow = ds.Tables["tabstudent"].NewRow();
      //给该记录赋值
      drow[0] = "19";
      drow[1] = "陈峰";
      drow[2] = "男";
      ds.Tables["tabstudent"].Rows.Add(drow);
      //提交更新
      sqld.Update(ds, "tabstudent");
      sqlconn.Close();
      sqlconn = null;
      Label1.Text = "增加成功";
    }
```

(4) 程序运行效果如图 8-15 所示。

图 8-15　DataAdapter_insert.aspx 的运行效果

8.4.4　使用 DataAdapter 对象删除数据库的数据

使用 DataAdapter 对象删除数据库数据的一般步骤为：首先建立数据库连接；然后利用数据库连接和 SELECT 语句建立 DataAdapter 对象；并建立 CommandBuilder 对象自动生成 DataAdapter 的 Command 命令；使用 DataAdapter 对象的 Fill 方法把 SELECT 语句的查询结果放在 DataSet 对象的数据表中；接下来将该数据表复制到 DataTable 对象中；最后实现对 DataTable 对象中数据的删除，并通过 DataAdapter 对象的 Update 方法向数据库提

交数据。

【例 8-10】演示如何使用 DataAdapter 对象删除符合条件的学生记录。

(1) 在 WebSite8 网站中添加一个名为 DataAdapter_delete.aspx 的网页。

(2) 向 DataAdapter_delete 页面添加 1 个 Label 控件，1 个 Button 控件，其中 Button 控件作为【删除】按钮。

(3) 双击设计视图中的【删除】按钮，添加如下后台代码。

```
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
• • •
protected void Button1_Click(object sender, EventArgs e)
{
    string sqlconnstr =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;
    SqlConnection sqlconn = new SqlConnection(sqlconnstr);
    DataSet ds = new DataSet();
    DataTable dtable;
    DataRowCollection coldrow;
    DataRow drow;
    sqlconn.Open();
    //建立 DataAdapter 对象
    SqlDataAdapter sqld = new SqlDataAdapter("select * from student", sqlconn);
    //建立 CommandBuilder 对象来自动生成 DataAdapter 的 Command 命令，否则就要自己编写
    //Insertcommand ,deletecommand , updatecommand 命令。
    SqlCommandBuilder cb = new SqlCommandBuilder(sqld);
     //用 Fill 方法返回的数据，填充 DataSet，数据表取名为 tabstudent
     sqld.Fill(ds, "tabstudent");
     dtable = ds.Tables["tabstudent"];
     coldrow = dtable.Rows;
     //逐行遍历，删除地址为空的记录
     for (int inti = 0; inti < coldrow.Count; inti++)
     {
         drow = coldrow[inti];
         if (drow["address"].ToString()== "")
             drow.Delete();
     }
     //提交更新
     sqld.Update(ds, "tabstudent");
     sqlconn.Close();
     sqlconn = null;
     Label1.Text = "删除成功";
}
```

(4) 程序运行效果如图 8-16 所示。

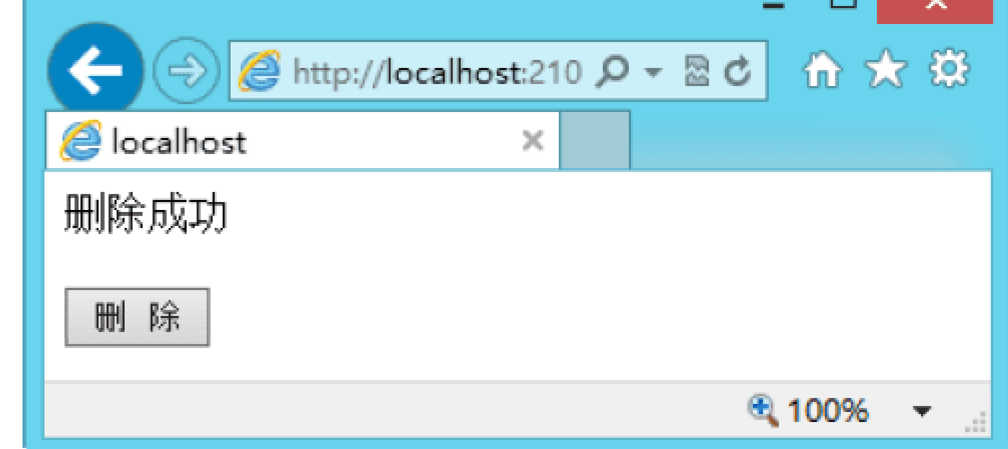


图 8-16　DataAdapter_delete.aspx 的运行效果

8.5　本章小结

本章全面讲述了 ADO.NET 数据访问技术。首先是 ADO.NET 技术的概述；然后介绍使用 Connection 对象连接数据库的方法；接着介绍使用 Command 对象执行数据库命令，包括查询、插入、删除、修改命令，以及数据库事务处理操作；最后介绍了使用 DataAdapter 对象执行数据库命令，同样包括了对数据库的增、删、改、查操作命令。通过本章的学习，读者应该能够掌握在 ASP.NET 中访问数据库的方法，以便能够根据需求熟练地对数据库进行增、删、改、查操作。

8.6　思考和练习

1. 简述 ADO.NET 的组成。
2. ADO.NET 提供了哪些操作数据库的对象？分别简述它们的功能。
3. 简述 DataSet 对象的结构。
4. ASP.NET 数据访问程序的开发过程有哪些步骤？
5. 如何通过 Command 对象实现对数据库数据的增、删、改、查操作？
6. 如何通过 DataAdapter 对象实现对数据库数据的增、删、改、查操作？

第9章　ASP.NET中的数据绑定

本章首先介绍单值绑定和列表控件的数据绑定过程，然后介绍 GridView 等复杂数据绑定控件的基本用法，主要涉及以下 3 种复杂数据绑定控件：GridView、DataList 和 FormView。本章将通过一系列实例逐步学习如何控制控件的显示效果和行为。

本章的学习目标：

- 掌握单值和列表控件的数据绑定。
- 掌握 GridView 的数据绑定方式、绑定列模板列的配置和用法以及数据的排序和分页。
- 掌握 DataList 和 FormView 的数据绑定和自定义样式的设置。

9.1　数据绑定概述

第 8 章学习了如何通过 ADO.NET 访问数据库，而这一章将着重学习如何利用 ASP.NET 提供的控件将数据呈现在页面上。大家知道，Web 系统的一个典型的特征是后台对数据的访问和处理与前台数据的显示分离，而前台显示是通过 HTML 来实现的。一种将数据呈现的最直接的方式是将需要显示的数据和 HTML 标记拼接成字符串并输出，但这种方案的缺点也是显而易见的，不但复杂而且难以重用，尤其是有大宗数据需要处理时。因此为了简化开发过程，ASP.NET 环境中提供了多种不同的服务器端控件来帮助程序员更快速高效地完成数据的呈现。这些用于数据呈现的 ASP.NET 控件，集成了常见的数据显示框架和数据处理功能，因而在使用时只需要设置某些属性，并将需要显示的数据交付给控件，控件就可以帮助用户按照固定的样式(如表格)或通过模板自定义样式将一系列数据呈现出来，并自动继承某些内置的数据处理功能，如排序、分页等，当然也可以通过编程定制或扩展控件的行为。这些控件就被称为数据绑定控件，而将数据交付给数据绑定控件的过程就被称为数据绑定。

数据绑定控件本质上依然是通过 HTML 来呈现数据的，只不过按照某种样式生成 HTML 框架并将数据填入其中的工作由控件自动完成了，一些复杂的数据绑定控件还提供了大量的功能帮助用户对数据进行进一步操作，如排序、过滤、新增、修改和删除等，因而使得数据呈现的过程变得简单而灵活。正因为如此，数据绑定控件的使用是 ASP.NET 编程中非常重要的一部分内容。

本节首先从简单的单值控件和列表控件讲起，然后通过多个例子详细地介绍最常用的一种数据绑定控件 GridView 的各个方面，最后介绍 DataList、FormView 和 Repeater 这 3

种用法相近的数据绑定控件。

9.2　单值和列表控件的数据绑定

数据绑定本质上是将数据在前台页面上呈现，ASP.NET 的页面元素可以简单地分成两类：HTML 标记和服务器控件，因此数据绑定实际上是在 HTML 标记中或服务器控件中设置要显示数据的过程。对于页面中的 HTML 标记，可以直接嵌入数据或绑定表达式来设置要显示的数据，而对于服务器控件来说，通常通过设置控件属性或指定数据源来完成数据的绑定，并控制其呈现的样式。常用的绑定表达式具有以下形式：<%#XXX%>，绑定表达式可以直接嵌入到前台页面代码中去，通常用于 HTML 标记中的数据显示或单值控件数据设置，如 Label、TextBox 等。而对于列表控件(如 DropDownList、CheckBoxList)，以及后面要着重介绍的复杂数据绑定控件则常采用设置数据源的方式完成数据呈现。

9.2.1　单值绑定

【例 9-1】演示了单值绑定的方法，其中通过绑定表达式和后台设置属性两种方式设置在 HTML 标记中和一个简单服务器控件中要显示的数据，对单值控件的数据设置方式是通用的，因此这里只以 TextBox 为例。

【例 9-1】演示如何进行单值绑定。

(1) 启动VS，新建一个名为WebSite9的ASP.NET网站，并在网站中添加一个名为SingleValueBinding.aspx的网页。新建数据库MyDatabase.mdf，将WebSite8中的MyDatabase.mdf拷贝到WebSite9相应的目录App_Data下。

(2) 返回到 StringValueBinding.aspx 页面，在<div/>标记中添加 1 个 Label 控件和 2 个 TextBox 控件，并修改页面代码如下。

```
    <div>
        <%# SingleValueBindingStr+"0" %>
         <asp:Label ID="Label1" runat="server" Text=" <%# SingleValueBindingStr+"1"
%>"></asp:Label>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <asp:TextBox ID="TextBox2" runat="server" Text="Label2"></asp:TextBox>
         </div>
```

(3) 修改页面后台代码如下。

```
    public partial class SingleValueBinding : System.Web.UI.Page
        {
            //在页面代码中将通过绑定表达式直接引用该成员
            public String SingleValueBindingStr = "单值绑定";
            protected void Page_Load(object sender, EventArgs e)
            {
                //页面的数据绑定方法，对于绑定表达式来说是关键的一步
                Page.DataBind();
```

```
            //通过在后台设置服务器控件属性来绑定数据
            this.TextBox1.Text = this.SingleValueBindingStr + "2";
            this.TextBox2.Text = this.SingleValueBindingStr + "3";
        }
    }
```

(4) 程序运行效果如图 9-1 所示。

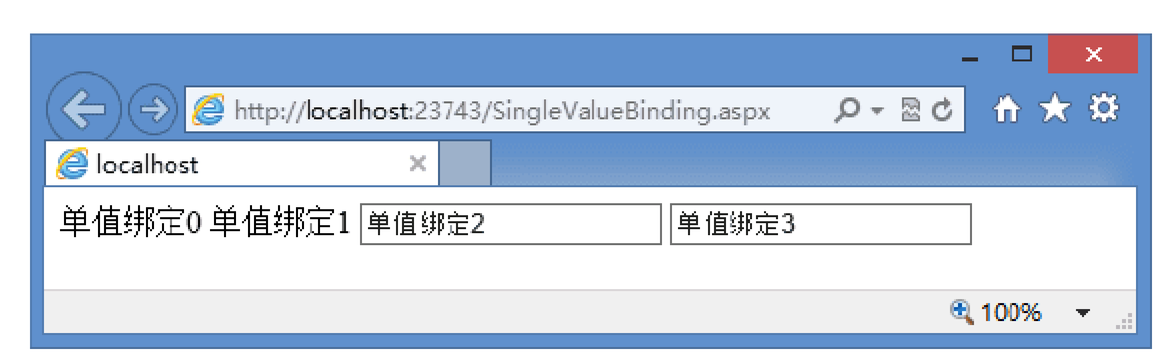


图 9-1　SingleValueBinding.aspx 运行效果

在【例 9-1】中，首先用绑定表达式<%# SingleValueBindingStr + "0" %>在 Div 标记中显示绑定值，然后将变量值绑定到页面的 Label 控件。最后通过在后台代码中将变量值绑定到 TextBox 控件。

这里需要注意绑定表达式的用法，在<%#%>标记中通过直接引用页面类中定义的公有数据成员 SingleValueBindingStr 构成表达式，这是因为 aspx 页面是从.cs 代码页中的类型继承过来的，而<%#%>标记的作用正是通过在前台显示代码中嵌入访问后台数据的表达式来完成数据绑定。这实际上是通过绑定表达式建立了后台代码与前台页面元素之间的联系，在输出页面流时，系统根据绑定表达式引用后台代码产生的数据计算表达式的值并插入到显示页面的合适位置。因而，在绑定表达式中不仅可以引用后台代码中的公有数据成员，而且可以引用其公有方法，有兴趣的读者可以自己试验一下。另外，通过表达式绑定数据实际上包含两个步骤：除了为 HTML 元素或服务器空间指定绑定表达式外，还需要在后台代码中显式调用控件的 DataBind()方法，否则绑定过程不能完成，将不能产生预期的显示效果，支持数据绑定的控件都提供了 DataBind()。所以在【例 9-1】后台代码的 Page_Load 方法中才必须包含 Page.DataBind()方法的调用，这里 Page.DataBind()方法会调用页面所有控件及其子控件的 DataBind()方法(包括 HTML 元素)。

对于服务器控件 TextBox1，直接在后台代码中设置了其 TextBox1.Text 属性值来完成数据绑定。这也是一种通用的方法，不仅是对单值控件，对于更复杂的数据绑定控件也是适用的。只不过当涉及多值绑定时，需要挨个设置各个显示项的相应属性，当项数较多时会比较繁琐，所以对于多值绑定更常用的方法是通过设置控件数据源来完成数据绑定。

9.2.2　列表控件的数据绑定

ASP.NET 中的列表控件有以下几种：DropDownList、ListBox、CheckBoxList、RadioButtonList、BulletedList 等。虽然这些控件呈现数据的样式和某些功能有所不同，但本质上都是以数据项列表的形式呈现和组织数据的集合，因此数据绑定的方式也很相似。可以通过编程方式为控件对象增加多个数据项，也可以直接在 VS 环境提供的图形界面中编辑要显示的数据项列表。但由于通常列表控件绑定的是一个数据集合，上述两种方式就比较繁琐了，所以对于列表控件的数据绑定来说，更常用的方式是指定数据源然后调用控

件的 DataBind()方法。

下面以 3 种具有代表性的列表控件 DropDownList、CheckBoxList、BulletedList 为例，演示如何通过设置数据源绑定数据。C#中提供的很多集合类都可以作为列表控件的数据源对象。一般来说，实现 IEnumerable、IListSource、IDataSource 和 IHierarchicalDataSource 的类都可以作为数据源。【例 9-2】通过 3 种不同的方式构成 3 种集合类对象作为 3 种列表控件的数据源。

【例 9-2】通过设置数据源绑定列表控件。

(1) 在 WebSite9 网站中新建一个名为 ListValueBinding.aspx 页面并在其中添加 3 种列表控件，如图 9-2 所示。其中，在页面中添加了如下控件。

- 一个 DropDownList，一个文本框用于显示选择项的值，设置其 AutoPostBack 属性为 True。
- 一个 CheckBoxList，一个确定按钮和一个文本框，控件每一个绑定项显示文本为学生姓名，而值为性别，当按下确定按钮后，将在文本框中显示 CheckBoxList 中所选姓名所对应的性别(键值对)。
- 一个用于显示链接列表的 BulletedList，每一个绑定项都描述一个键值对，代表指向某个网站的链接。显示文本为超链接形式的网站名称，而值为网站 url，因此需要设置控件的 DisplayMode="HyperLink"，并设置 Target="_blank"，代表单击后目标 url 将在新窗口中打开。

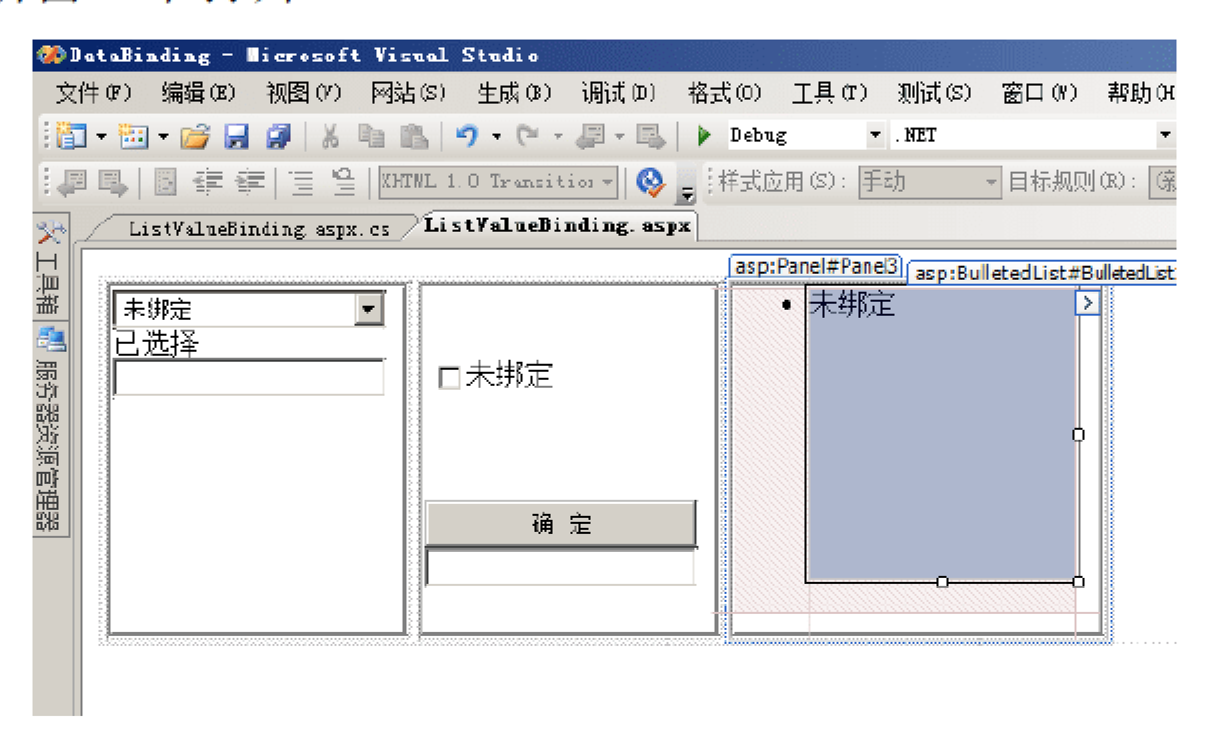


图 9-2　ListValueBinding 设计页面

页面代码如下。

```
    <div>
                <table style="width: 480px;">
                    <tr>
                        <td>
                            <asp:Panel ID="Panel1" runat="server" Height="190px" Width="160px"
                                BorderStyle="Groove">
                                <asp:DropDownList ID="DropDownList1" runat="server"
AutoPostBack="True"
                                    Height="53px" Width="150px"
OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
                                </asp:DropDownList>
                                <br />
```

```
                        <asp:Label ID="Label1" runat="server" Text="已选择"
Width="92%"></asp:Label>
                        <br />
                        <asp:TextBox ID="TextBox1" runat="server"
Width="150px"></asp:TextBox>
                        <br />
                    </asp:Panel>
                </td>
                <td>
                    <asp:Panel ID="Panel2" runat="server" Height="190px" Width="160px"
                        BorderStyle="Groove" >
                        <asp:CheckBoxList ID="CheckBoxList1" runat="server"
Height="98px" Width="100%">
                            <asp:ListItem>未绑定</asp:ListItem>
                        </asp:CheckBoxList>
                        <br />
                        <asp:Button ID="Button1" runat="server" Text="确　定"
Width="150px" OnClick="Button1_Click" />
                        <br />
                        <asp:TextBox ID="TextBox2" runat="server" Width="150px"
></asp:TextBox>
                        <br />
                    </asp:Panel>
                </td>
                <td>
                   <asp:Panel ID="Panel3" runat="server" Height="190px" Width="200px"
                        BorderStyle="Groove">
                        <asp:BulletedList ID="BulletedList1" runat="server"
                            Height="160px" Target="_blank" Width="73%"
BulletStyle="Disc">
                        </asp:BulletedList>
                   </asp:Panel>
                </td>
            </tr>
        </table>
    </div>
```

(2) 在后台页面类中添加如下代码。

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebSite9
{
    public partial class ListValueBinding : System.Web.UI.Page
    {
        //定义三种数据源
```

```
            //定义并初始化字符串数组
            String[] DataSourceForDDL = new String[] { "张小兵", "李明", "陈飞" };
            //定义哈希表
            Hashtable DataSourceForCBL = new Hashtable(3);
            //定义 ArrayList
            ArrayList DataSourceForBL = new ArrayList();
            protected void Page_Load(object sender, EventArgs e)
            {
                if (!IsPostBack)
                {
                    //初始化哈希表 DataSourceForCBL
                    this.DataSourceForCBL.Add("徐明明", "男");
                    this.DataSourceForCBL.Add("韩旭", "女");
                    this.DataSourceForCBL.Add("陈静", "女");
                    //初始化 DataSourceForBL
                    this.DataSourceForBL.Add(new KeyValueClass("百度",
"http://www.baidu.com"));
                    this.DataSourceForBL.Add(new KeyValueClass("CSDN",
"http://www.CSDN.net"));
                    //为 DropDownList 绑定数据
                    this.DropDownList1.DataSource = this.DataSourceForDDL;
                    this.DropDownList1.DataBind();
                    //完成绑定后在 DropDownList 中第一个位置插入一个数据项
                    this.DropDownList1.Items.Insert(0, "请选择");
                    //为 CheckBoxList 绑定数据
                    this.CheckBoxList1.DataSource = this.DataSourceForCBL;
                    //由于哈希表中存储一个键值对的集合并希望在 CheckBoxList
                   //中处理键值对，因此设定数据源后还需设定 DataTextField 和 DataValueField
属性
                    this.CheckBoxList1.DataTextField = "key";
                    this.CheckBoxList1.DataValueField = "value";
                    this.CheckBoxList1.DataBind();
                    //为 BulletedList 绑定数据
                    this.BulletedList1.DataSource = this.DataSourceForBL;
                    this.BulletedList1.DataTextField = "Name";
                    this.BulletedList1.DataValueField = "Url";
                    this.BulletedList1.DataBind();
                }
            }
        }
    }
```

代码中定义了 3 种集合类对象并进行了初始化：字符串数组 DataSourceForDDL 用作 DropDownList 的数据源；哈希表 DataSourceForCBL 用作 CheckBoxList；ArraryList 对象 DataSourceForBL 用作 BulletedList 的数据源。3 种列表控件的数据项对象都具有以下两种属性：DataTextField 和 DataValueField，其中只有 DataTextField 的值会被呈现，因此它们既可以处理单值集合也可以处理键值对数据的集合。

注意：

当处理键值对时，在指定数据源后，要设置上述两个属性以指定数据源中对应于键和

值的数据域。

(3) 然后再添加一个帮助类 KeyValueClass，其中可以存储一个键值对，用于初始化 DSForBL 对象。

```
/// <summary>
/// 帮助类 KeyValueClass 的定义，其对象存储一个 WebSiteName：WebSiteUrl
/// 的键值对，被添加至作为数据源的 ArrayList 中
/// </summary>
using System;
public class KeyValueClass
{
    private String WebSiteName;
    private String WebSiteUrl;
    public String Name
    {
        get { return WebSiteName; }
        set { WebSiteName = value; }
    }
    public String Url
    {
        get { return WebSiteUrl; }
        set { WebSiteUrl = value; }
    }
    public KeyValueClass(String name, String url)
    {
        this.WebSiteName = name;
        this.WebSiteUrl = url;
    }
}
```

(4) 运行后结果如图 9-3 所示。

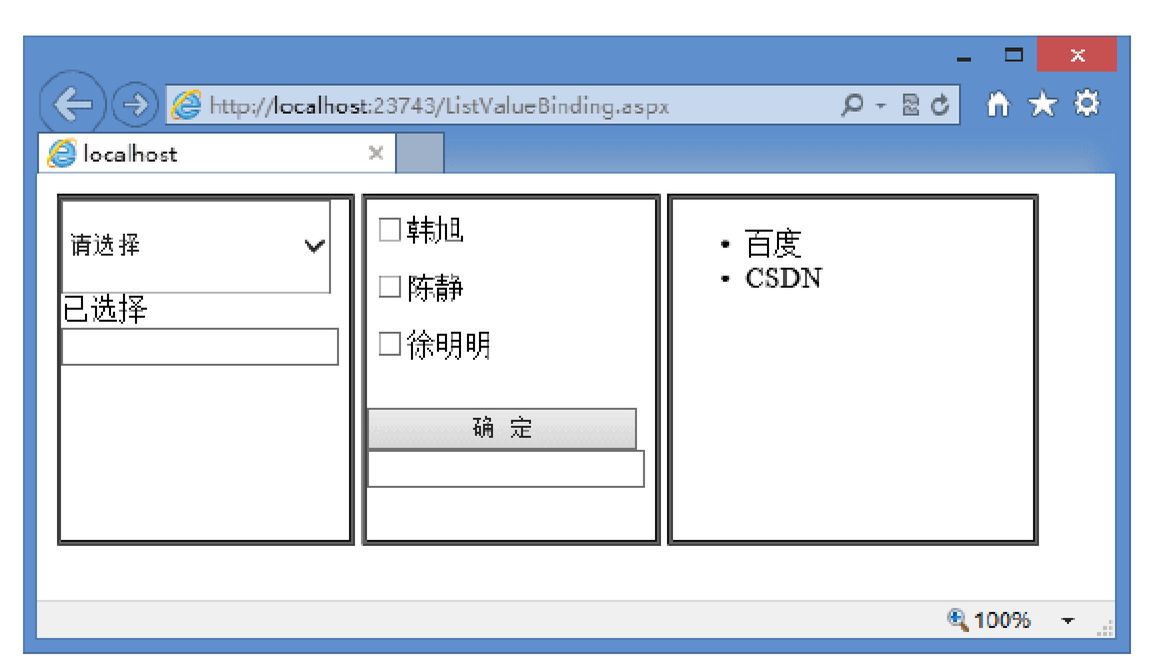


图 9-3　数据绑定后的列表控件

当数据绑定完成后就可以通过控件提供的各种事件定制其行为，对数据进行进一步处理。

(5) 为 DropDownList 和【确定】按钮添加事件处理代码如下。

```
protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
    {
```

```
            //清除上次显示内容
            TextBox1.Text = "";
            //在文本框中显示所选列表项
            TextBox1.Text = DropDownList1.SelectedValue;
        }
    protected void Button1_Click(object sender, EventArgs e)
        {
            TextBox2.Text = "";
            //循环遍历 CheckBoxList 中每个项，如果已选择在文本框中显示选中项显示文本和值
            foreach (ListItem li in CheckBoxList1.Items)
            {
                if (li.Selected) TextBox2.Text += li.Text + ":" + li.Value + ",";
            }
        }
```

(6) 运行效果如图 9-4 所示。



图 9-4　事件处理运行效果

9.3　GridView 控件

GridView 是一个功能强大的数据绑定控件，主要用于以表格形式呈现、编辑关系数据集。对应于关系数据集的结构，GridView 控件以列为单位组织其所呈现的数据，除了普通的文本列，还提供多种不同的内置列样式，如按钮列、图像列、复选框形式的数据列等，可以通过设置 GridView 控件的绑定列属性以不同的样式呈现数据，或通过模板列自定义列的显示样式。

在数据绑定时通常将访问关系数据库得到的结果集作为 GridView 控件的数据源，GridView 控件对其所呈现的数据集提供内置的编辑、修改、更新、删除以及分页和排序功能，但是要使用控件的内置数据处理功能，需要使用 ASP.NET 提供的数据源控件(如 SqlDataSource 和 ObjectDataSource)，否则需要手动编写事件处理程序来实现相应功能。虽然采用数据源控件来连接数据库并处理数据更加方便，但手动编写代码却更加灵活，并且在编写代码的过程中可以更深入地了解 GridView 控件的运行方式，因而更具有参考意义。所以本章的例子将采用查询数据库得到的 DataTable 对象作为控件数据源，然后通过编写事件程序的方式来实现数据处理功能，而在下一节中采用数据源控件绑定 FormView 控件

并使用内置的数据处理功能。

本节将从以下 3 个方面介绍 GridView 控件的使用：

- 基本的数据绑定方式操作；
- 各种绑定列和模板列的灵活运用；
- 数据的分页和排序功能的实现。

9.3.1　GridView 的数据绑定

【例 9-3】将演示如何进行 GridView 的数据绑定。其基本的数据绑定方式与列表控件类似，首先设置数据源，后调用 DataBind()方法。本例在一个 GridView 控件中呈现关系数据集，因此需要访问数据库获取作为控件数据源的结果集，数据库沿用前一章的 student 数据库。数据库创建和访问方式在前一章已经详细介绍过，这里就不再赘述。

【例 9-3】为 GridView 控件绑定数据源。

(1) 在 WebSite9 网站中新建一个名为 GridViewBingding_1.aspx 的页面，并在页面上添加一个 GridView 控件，如图 9-5 所示。

Column0	Column1	Column2
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc

Label

图 9-5　GridViewBingding_1.aspx 设计页面

对应页面代码如下。

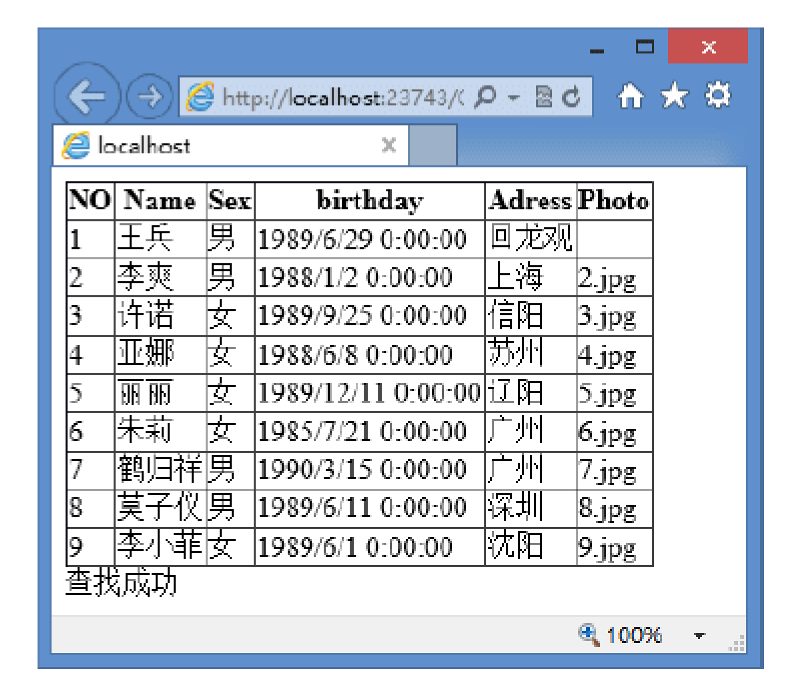
```
<div>
    <asp:GridView ID="GridView1" runat="server"> </asp:GridView>
    <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label><br />
</div>
```

(2) 为页面后台类添加数据绑定代码如下。

```
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
……
namespace WebSite9
{
    public partial class GridViewBingding_1 : System.Web.UI.Page
    {   //编写以下代码
        protected void Page_Load(object sender, EventArgs e)
        {
            //查询 student 数据库获取结果集 ds
            string sqlconnstr = ConfigurationManager.ConnectionStrings
["ConnectionString"].ConnectionString;
            DataSet ds = new DataSet();
            using (SqlConnection sqlconn = new SqlConnection(sqlconnstr))
            {
                SqlDataAdapter sqld = new SqlDataAdapter("select * from student", sqlconn);
                sqld.Fill(ds, "tabstudent");
            }
            //以数据集中名为 tabstudent 的 DataTable 作为数据源，为控件绑定数据
            GridView1.DataSource = ds.Tables["tabstudent"].DefaultView;
```

```
                GridView1.DataBind();
                //label 中显示运行状态
                Label1.Text = "查找成功";
            }
        }
    }
```

(3) 运行效果如下图 9-6 所示。



NO	Name	Sex	birthday	Adress	Photo
1	王兵	男	1989/6/29 0:00:00	回龙观	
2	李爽	男	1988/1/2 0:00:00	上海	2.jpg
3	许诺	女	1989/9/25 0:00:00	信阳	3.jpg
4	亚娜	女	1988/6/8 0:00:00	苏州	4.jpg
5	丽丽	女	1989/12/11 0:00:00	辽阳	5.jpg
6	朱莉	女	1985/7/21 0:00:00	广州	6.jpg
7	鹤归祥	男	1990/3/15 0:00:00	广州	7.jpg
8	莫子仪	男	1989/6/11 0:00:00	深圳	8.jpg
9	李小菲	女	1989/6/1 0:00:00	沈阳	9.jpg

图 9-6　GridViewBingding_1.aspx 数据绑定效果

9.3.2　设定 GridView 的绑定列和模板列

由【例 9-3】可以看出，GridView 的数据绑定方式操作非常简单，只用几句简单的代码就可以将数据集以表格形式呈现出来，但这种方式的呈现效果很简陋。实际上，可以通过设置 GridView 控件的绑定列属性使其呈现不同的列样式，实现数据的编辑和修改，或编辑模板列定制所需的列样式和功能。

下面【例 9-4】将演示如何为 GridView 设置绑定列、调整数据呈现效果、实现数据的编辑和修改功能以及如何通过定义列模板使其呈现自定义样式。在开始之前先来看看 GridView 提供了哪几种内置的绑定列样式，如何在 VS 中设置控件及如何定义模板列样式。

在 VS 中可以通过边界任务面板进行列的配置，如图 9-7 所示，单击 GridView 右上角的小箭头打开面板。

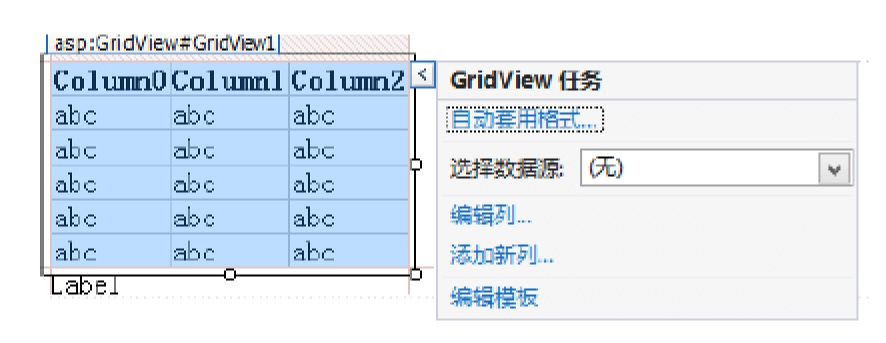


图 9-7　GridView 便捷任务面板

可以看到，面板上包含【自动套用格式】选项，通过【自动套用格式】可以为 GridView 应用一些内置的表格呈现样式。这里着重介绍【编辑列】选项和【编辑模板】选项。其中，【编辑列】选项用于设置表格的绑定列数据性，而【编辑模板】选项用于编辑模板列中的显示项的样式。单击图 9-7 的“编辑列”选项打开设置 GridView 列样式的“字段”对话框，如图 9-8 所示。

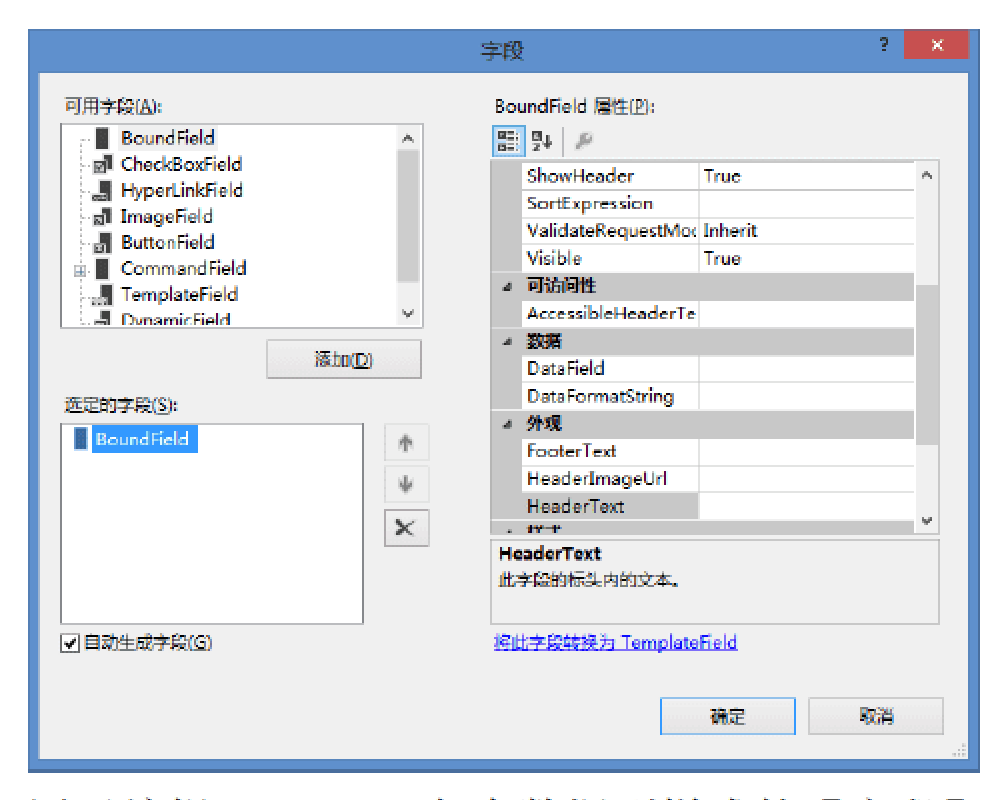


图 9-8　用于编辑 GridView 各个数据列样式的【字段】对话框

图 9-8 中左上角【可用字段】列表列出了可用的绑定列类型，单击【添加】按钮就可以设置 GridView 控件中显示的列及其类型。共有 8 种类型，如下所示。

- BoundField：以文字形式呈现数据的普通绑定列类型。
- CheckBoxField：以复选框形式呈现数据，绑定到该类型的列数据应该具有布尔值。
- HyperLinkField：以链接形式呈现数据，绑定到该类型的列数据应该是指向某个网站或网上资源的地址。
- ImageField：以图片形式呈现数据。
- ButtonField：按钮列，以按钮的形式呈现数据或进行数据的操作。例如删除记录的按钮列。
- CommandField：系统内置的一些操作按钮列，可以实现对记录的编辑、修改、删除等操作。
- TemplateField：模板列中对各个数据项并不给出预设的显示样式，而是通过自定义的方式在项模板中设置数据项的显示方式。

在实际应用的时候，可以根据需要显示的数据类型，选择要绑定的列类型并设置其映射到数据集的字段名称和呈现样式(设置绑定列后 GridView 中将只显示映射列数据，否则系统将默认以 BoundField 类型显示数据源表中的所有列)。例如，【例 9-3】中作为数据源的 student 表数据，如果想在 GridView 中以 BoundField 类型显示其中字段名为“姓名”的列，可以作如下设置，在图 9-8 所示字段编辑窗口中添加一个 BoundField 列，如图 9-9 所示。

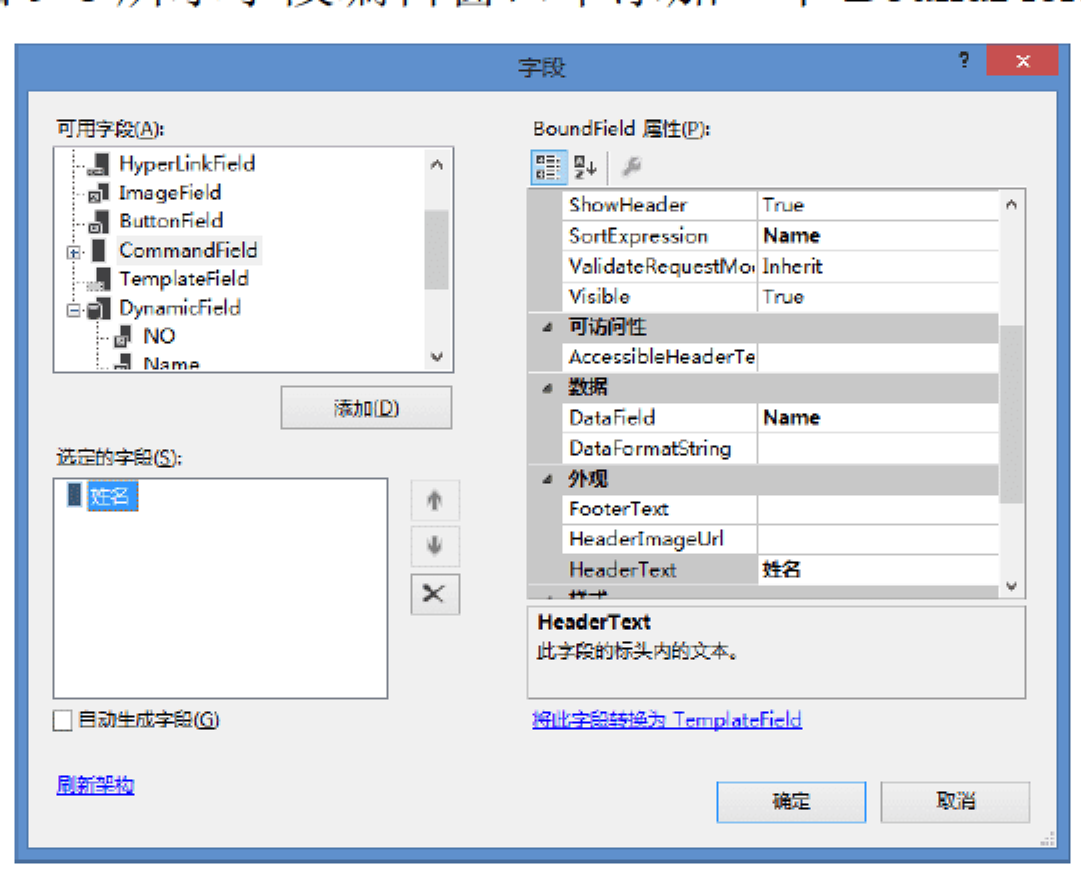


图 9-9　为 GridView 添加绑定列

在右方字段属性编辑框中设置 DataField 数据性为 Name，其中 Name 对应于作为数据源的 student 表中的 Name 字段，通过该属性完成显示列与源之间的数据映射，而 HeaderText 属性表示该字段呈现在 GridView 中时的表头名称，这里设置为“姓名”。在属性编辑框中还可以设置列的显示外观或行为等其他属性，这里就不再赘述。

通过类似的方式就可以为 GridView 控件添加其他类型的绑定列，这里 CommandField 的使用方式稍有特殊。通过 CommandField 类型，并配合事件处理程序就可以在 GridView 中完成数据的编辑、修改、插入等等操作。添加并设置 CommandField 类型的方式，展开 CommandField，如图 9-10 所示。

可见 CommandField 有 3 种类型可以选择，不同的类型意味着在 CommandField 列显示不同的命令按钮，如选择【编辑、更新、取消】，列样式如图 9-11 所示。

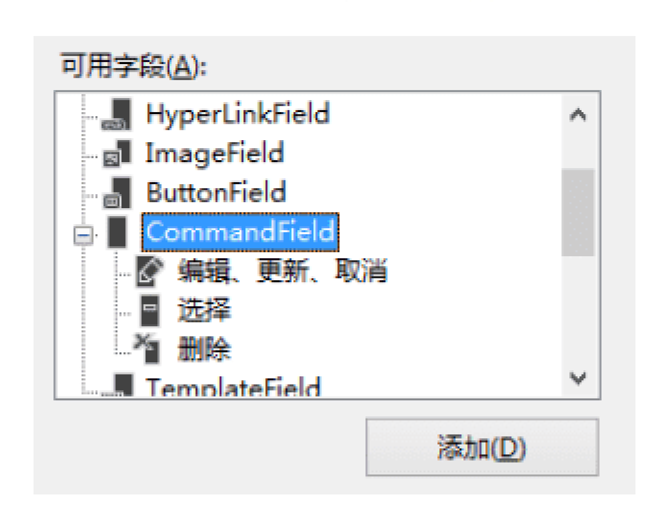


图 9-10　CommandField 类型

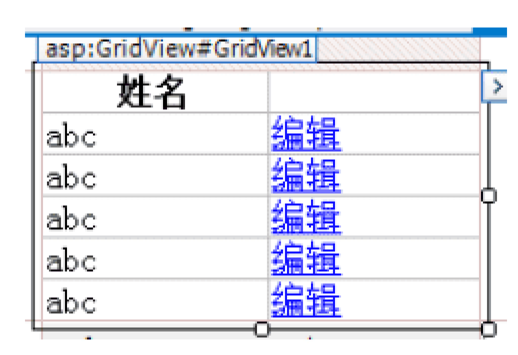


图 9-11　添加【编辑、更新、取消】列

运行时单击【编辑】按钮，列中的编辑按钮将会被替换为两个按钮【更新、取消】，因此列的运行时实际上包含了 3 个命令按钮，单击按钮所发生的行为需要通过设置相应的事件程序完成，由于 CommandField 类型是一种控件内置的用于编辑数据的绑定类型，因此其事件在 GridView 控件的属性窗口中设置，GridView 控件的属性窗口和选择事件列表如图 9-12 所示。

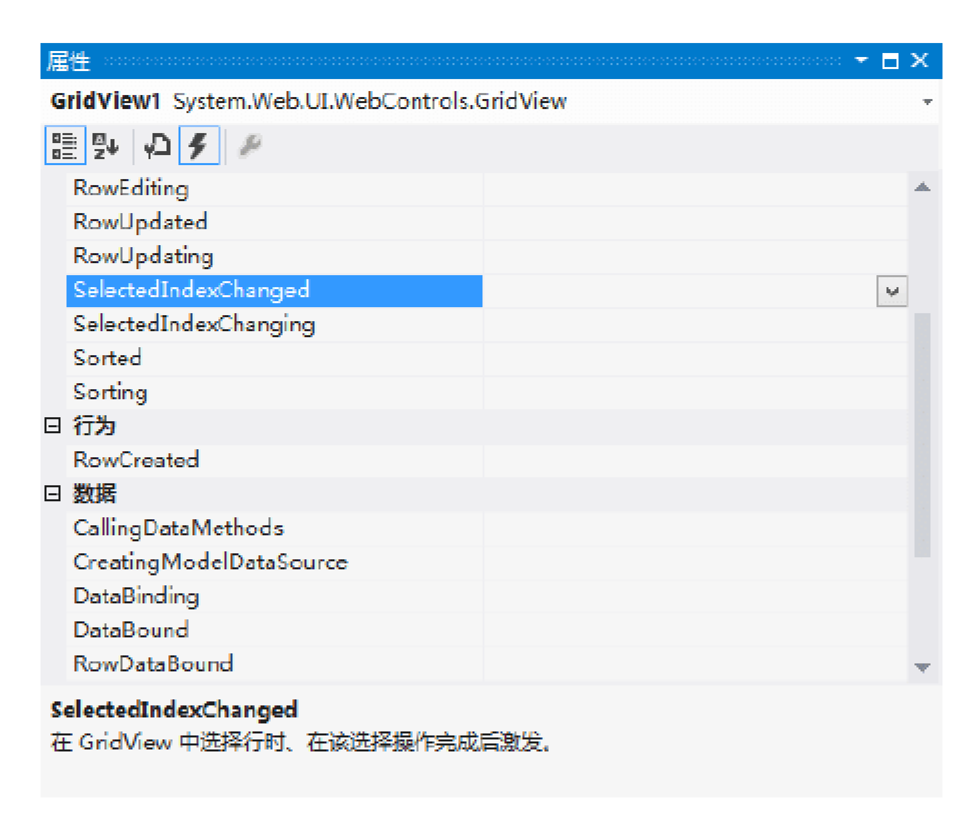


图 9-12　GridView 的事件编辑窗口

其中的 RowEditing、RowUpdating、RowDeleting、RowCancelingEdit 事件分别在编辑、更新、删除、取消按钮被单击时触发。通过为这些事件添加相应的处理程序就可以完成数据的编辑和修改功能。

对于 TemplateField 类型，需要先编辑模板来定义列中各个项的显示样式，然后根据自定义模板绑定模板列，系统将根据模板中定义的样式呈现数据。GridView 中自定义模板的方式与 DataList 或 Repeater 控件的模板定义方式类似，放在后续章节介绍。

【例 9-4】演示为 GridView 设置绑定列来显示 student 表数据的完整过程，【例 9-4】中将包含 4 个 BoundField 列，1 个 CommandField 列和 1 个 ButtonField 列，这里将为各个命令按钮添加事件处理程序完成数据的编辑、更新和删除功能。

【例 9-4】演示为 GridView 控件设置绑定列。

(1) 在 WebSite9 网站中新建一个名为 GridViewBingding_2.aspx 的页面，在页面上添加一个 GridView 控件，为其添加如下绑定列并设置数据映射，如图 9-13 所示。

- 学号列：BoundField 类型，绑定字段 no。
- 姓名列：BoundField 类型，绑定字段 name。
- 出生日期列：BoundField 类型，绑定字段 birth。
- 地址列：BoundField 类型，绑定字段 address。
- 编辑列：CommandField 类型，子类型为编辑、更新、取消，实现数据的编辑和更新。
- 删除按钮列：ButtonField 类型，实现记录的删除。

页面设计效果如图 9-14 所示。

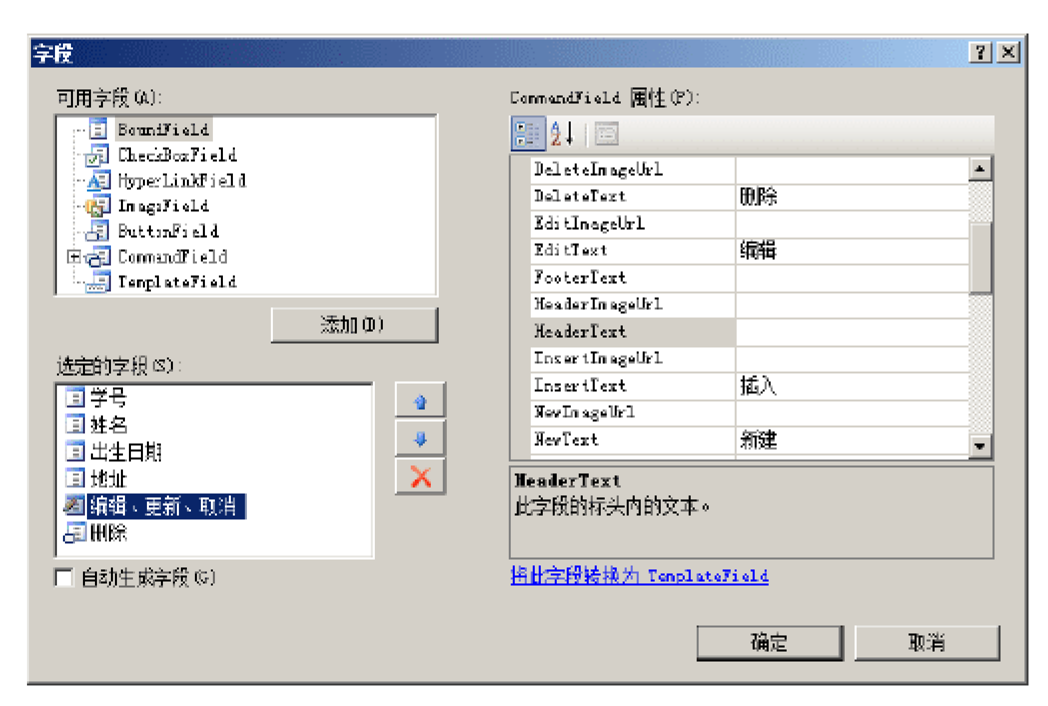


图 9-13　为 GridView 控件添加并设置绑定列

GridViewBingding_2.aspx

学号	姓名	出生日期	地址		
数据绑定	数据绑定	数据绑定	数据绑定	编辑	删除
数据绑定	数据绑定	数据绑定	数据绑定	编辑	删除
数据绑定	数据绑定	数据绑定	数据绑定	编辑	删除
数据绑定	数据绑定	数据绑定	数据绑定	编辑	删除
数据绑定	数据绑定	数据绑定	数据绑定	编辑	删除

图 9-14　GridViewBingding_2.aspx 设计页面

页面代码如下。

```
    <div>
            <asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
OnRowCancelingEdit="GridView1_RowCancelingEdit"
                OnRowEditing="GridView1_RowEditing"
OnRowUpdating="GridView1_RowUpdating"
                DataKeyNames="no" OnRowDeleting="GridView1_RowDeleting" Height="185px"
                Width="478px">
                <Columns>
                    <asp:BoundField DataField="No" HeaderText="学号" ReadOnly="True" />
                    <asp:BoundField DataField="Name" HeaderText="姓名" />
                    <asp:BoundField DataField="birthday" HeaderText="出生日期" />
                    <asp:BoundField DataField="Address" HeaderText="地址" />
                    <asp:CommandField InsertVisible="False" ShowEditButton="True" />
                    <asp:ButtonField ButtonType="Button" CommandName="delete" Text="删除" />
                </Columns>
            </asp:GridView>
     </div>
```

(2) 在页面后台类中添加如下代码，以进行数据绑定。

```
    using System.Configuration;
    using System.Data;
    using System.Data.SqlClient;
    ……
      protected void Page_Load(object sender, EventArgs e)
        {
            if (!Page.IsPostBack) bindgrid();
        }
        void bindgrid()
            {
                //查询 student 数据库获取结果集 ds
                string sqlconnstr =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;
                DataSet ds = new DataSet();
                using (SqlConnection sqlconn = new SqlConnection(sqlconnstr))
                {
```

```
                    SqlDataAdapter sqld = new SqlDataAdapter("select No,Name,birthday,Address
from student", sqlconn);
                    sqld.Fill(ds, "tabstudent");
                }
                //以数据集中名为 tabstudent 的 DataTable 作为数据源，为控件绑定数据
                GridView1.DataSource = ds.Tables["tabstudent"].DefaultView;
                GridView1.DataBind();
            }
```

(3) 为命令按钮列绑定事件处理方法，如图 9-15 所示。

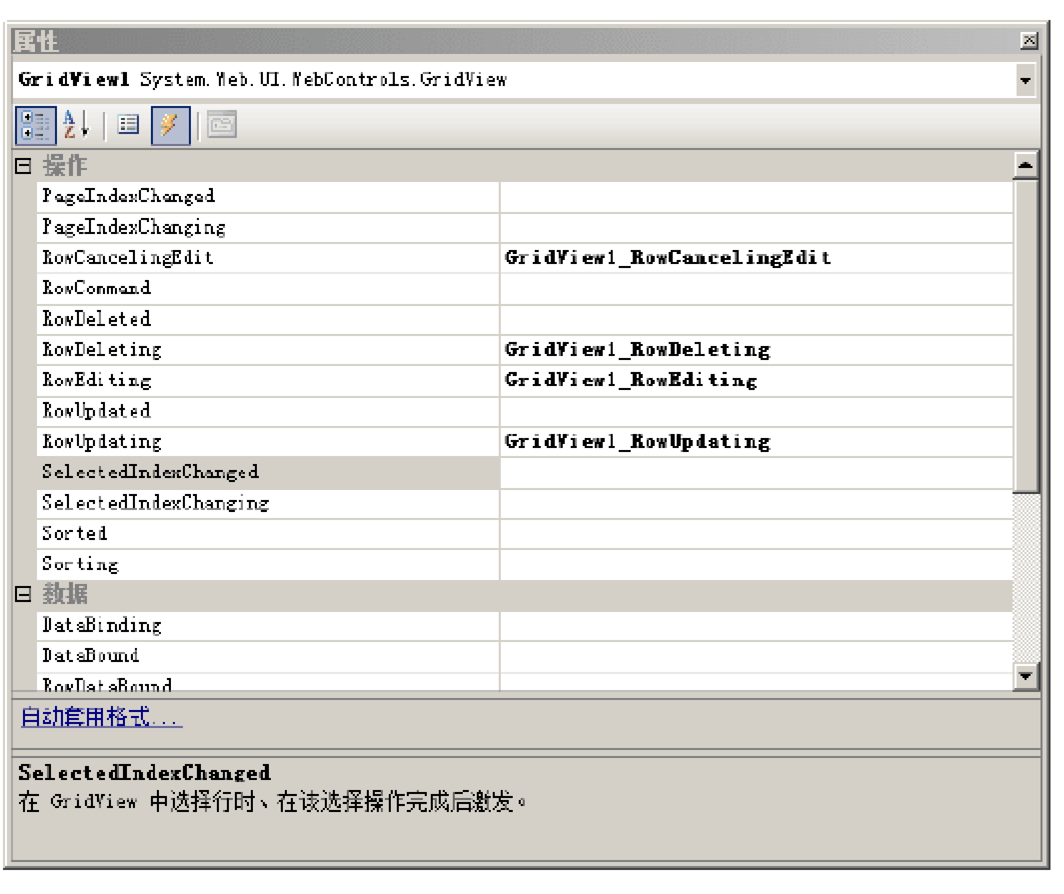


图 9-15　为命令事件设置事件处理方法

(4) 在页面后台事件处理方法中添加事件处理代码如下。

```
        protected void GridView1_RowEditing(object sender, GridViewEditEventArgs e)
            {
                GridView1.EditIndex = e.NewEditIndex;
                bindgrid();
            }
            protected void GridView1_RowUpdating(object sender, GridViewUpdateEventArgs e)
            {
                string sqlconnstr =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString; ;
                SqlConnection sqlconn = new SqlConnection(sqlconnstr);
                //提交行修改
                sqlconn.Open();
                SqlCommand Comm = new SqlCommand();
                Comm.Connection = sqlconn;
                Comm.CommandText = "update student set
Name=@Name,birthday=@birthday,Address=@Address where No=@No";
                Comm.Parameters.AddWithValue("@No",
GridView1.DataKeys[e.RowIndex].Value.ToString());
                Comm.Parameters.AddWithValue("@Name",
((TextBox)GridView1.Rows[e.RowIndex].Cells[1].Controls[0]).Text);
                Comm.Parameters.AddWithValue("@birthday",
((TextBox)GridView1.Rows[e.RowIndex].Cells[2].Controls[0]).Text);
                Comm.Parameters.AddWithValue("@Address",
((TextBox)GridView1.Rows[e.RowIndex].Cells[3].Controls[0]).Text);
```

```
                Comm.ExecuteNonQuery();
                sqlconn.Close();
                sqlconn = null;
                Comm = null;
                GridView1.EditIndex = -1;
                bindgrid();
            }
            protected void GridView1_RowCancelingEdit(object sender,
GridViewCancelEditEventArgs e)
            {
                GridView1.EditIndex = -1;
                bindgrid();
            }
            protected void GridView1_RowDeleting(object sender, GridViewDeleteEventArgs e)
            {
                //设置数据库连接
                string sqlconnstr =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString; ;
                SqlConnection sqlconn = new SqlConnection(sqlconnstr);
                sqlconn.Open();
                //删除行处理
                String sql = "delete from student where No='" +
GridView1.DataKeys[e.RowIndex].Value.ToString() + "'";
                SqlCommand Comm = new SqlCommand(sql, sqlconn);
                Comm.ExecuteNonQuery();
                sqlconn.Close();
                sqlconn = null;
                Comm = null;
                GridView1.EditIndex = -1;
                bindgrid();
            }
```

(5) 页面运行效果如图 9-16 所示。

(6) 单击【编辑】按钮后出现编辑和更新界面，如图 9-17 所示。

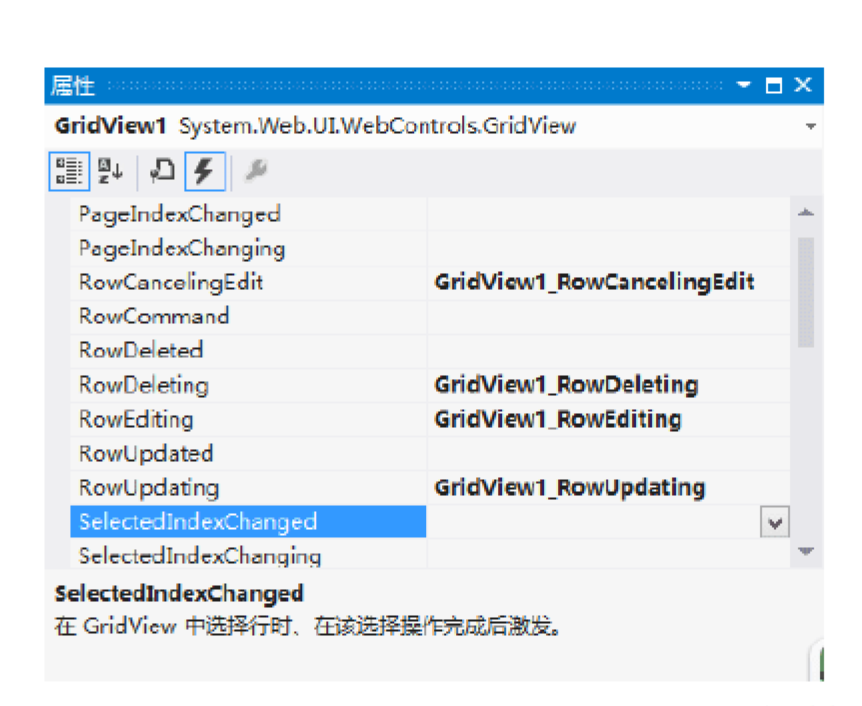


图 9-16　GridViewBingding_2.aspx 运行效果



图 9-17　GridViewBingding_2.aspx 的编辑效果

9.3.3　GridView 的排序

GridView 控件提供了用于实现排序功能的接口，通过设置相关属性并实现排序事件的处理代码就可以完成排序功能。这里将以【例 9-4】提供的界面的基础上实现排序功能。

【例 9-5】通过 GridView 控件实现数据记录排序功能。

(1) 在【例 9-4】中 GridViewBingding_2.aspx 页面中设置 GridView 控件属性 AllowSorting=True，如图 9-18 所示。

除了 AllowSorting 属性，还必须设置作为排序关键字的列的 SortExpression 属性，这是因为 GridView 中可以包含按钮列，按钮列一般并不映射到某个数据字段，而排序必须以某个字段作为排序关键字才能完成。

(2) 在 GridView 控件的便捷任务面板中打开“编辑列”对话框，选择可以作为排序关键字的列，设置其 SortExpression 属性为排序字段名，如图 9-19 所示。

这时作为排序关键字的列名变为超链接样式，如图 9-20 所示。

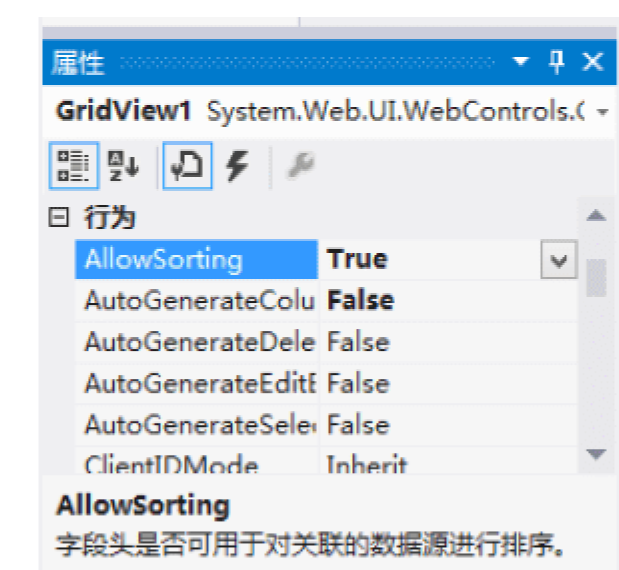


图 9-18　设置 AllowSorting 属性

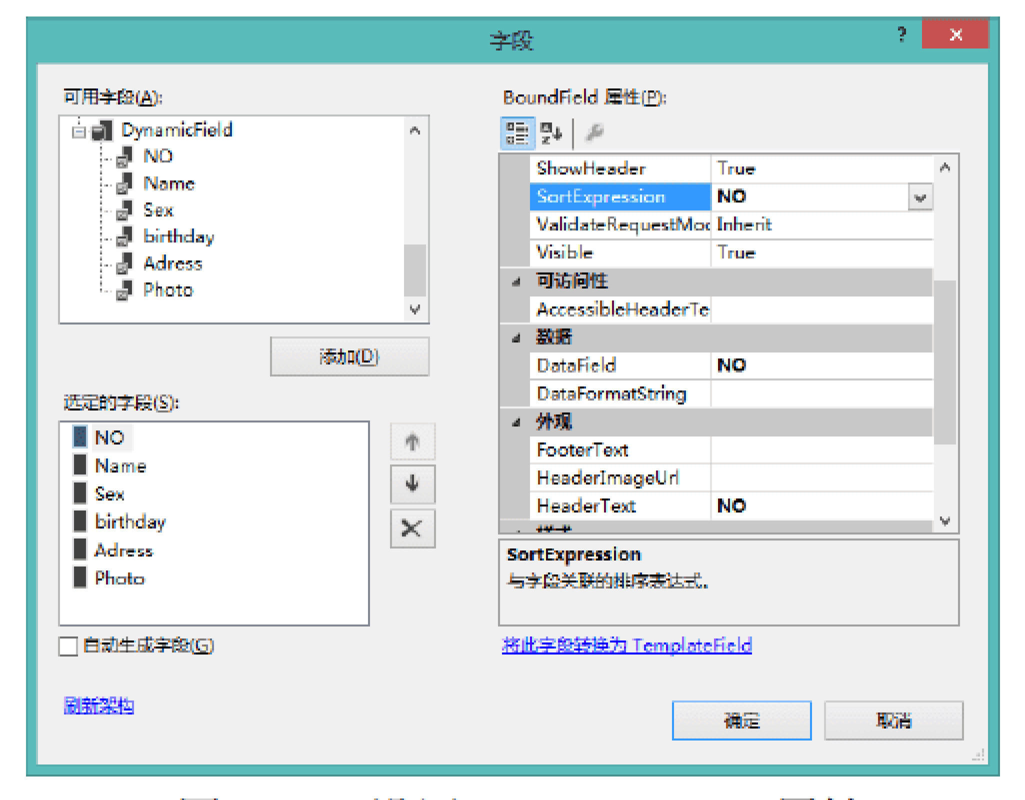


图 9-19　设置 SortExpression 属性



图 9-20　设置排序属性后的控件样式

(3) 为 GridView 控件设置排序事件处理方法，如图 9-21 所示。

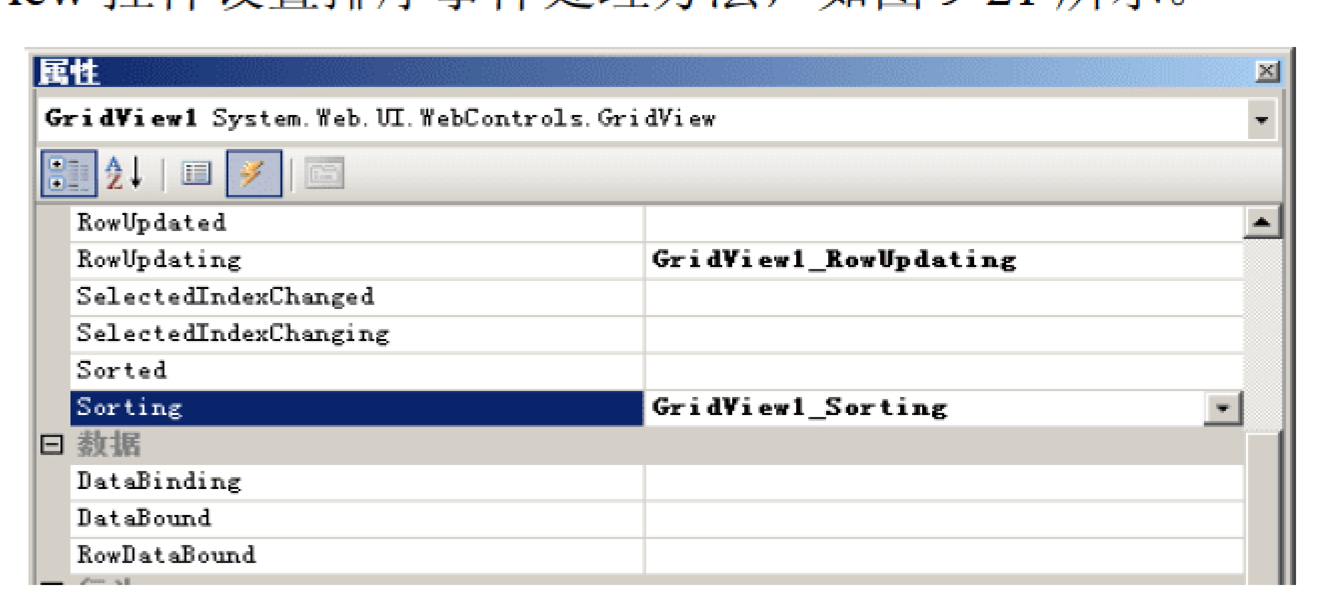


图 9-21　为控件设置排序事件处理方法

GridView 的排序功能通过响应排序事件在后台生成已排序的数据源后重新绑定数据来完成，因此需要事件相应代码中需要获取排序字段名和排序方式(升序、降序)，然后据此对数据源进行排序后重新绑定数据。

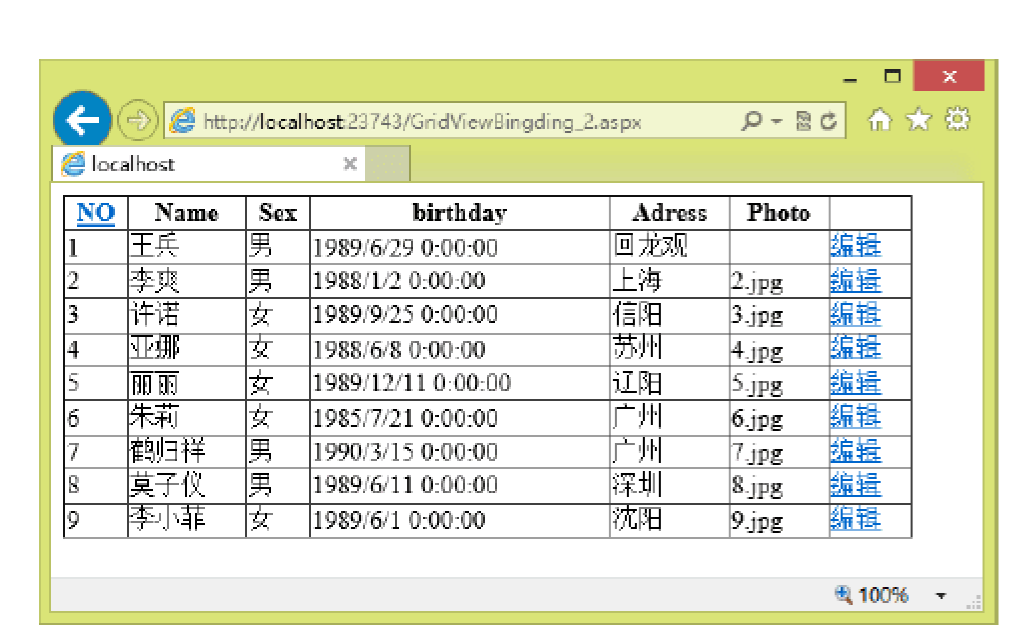
(4) 为排序事件处理方法添加代码如下，代码中用一个 ViewState["SortDirection"]来记录当前的排序顺序，用一个 ViewState["SortExpression"]记录作为排序关键字的字段名，然后重新绑定数据。

```
protected void GridView1_Sorting(object sender, GridViewSortEventArgs e)
{
    if(ViewState["SortDirection"] == null) ViewState["SortDirection"] = "DESC";
    if (ViewState["SortDirection"] .ToString() == "ASC")
        ViewState["SortDirection"] = "DESC";
    else
        ViewState["SortDirection"] = "ASC";
    ViewState["SortExpression"] = e.SortExpression;
    this.DataBind();
}
```

修改 DataBind()代码如下，使其根据 ViewState["SortDirection"]的值生成排序后的 DataView 对象作为数据源。

```
    void bindgrid()
        {
            string sqlconnstr =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString; ;
            DataSet ds = new DataSet();
            using (SqlConnection sqlconn = new SqlConnection(sqlconnstr))
            {
                SqlDataAdapter sqld = new SqlDataAdapter("select * from student", sqlconn);
                sqld.Fill(ds, "tabstudent");
            }
            //判断是否已经进行排序，如果是则按照 ViewState 中存储的信息生成排序后的
DataView 对象
            if (ViewState["SortDirection"] == null)
                GridView1.DataSource = ds.Tables["tabstudent"].DefaultView;
            else
            {
                DataView SortedDV = new DataView(ds.Tables["tabstudent"]);
                SortedDV.Sort = ViewState["SortExpression"].ToString() + " " +
ViewState["SortDirection"].ToString();
                GridView1.DataSource = SortedDV;
            }
            GridView1.DataBind();
        }
```

(5) 排序效果如图 9-22 所示。



NO	Name	Sex	birthday	Adress	Photo	
1	王兵	男	1989/6/29 0:00:00	回龙观		编辑
2	李爽	男	1988/1/2 0:00:00	上海	2.jpg	编辑
3	许诺	女	1989/9/25 0:00:00	信阳	3.jpg	编辑
4	亚娜	女	1988/6/8 0:00:00	苏州	4.jpg	编辑
5	丽丽	女	1989/12/11 0:00:00	辽阳	5.jpg	编辑
6	朱莉	女	1985/7/21 0:00:00	广州	6.jpg	编辑
7	鹤归祥	男	1990/3/15 0:00:00	广州	7.jpg	编辑
8	莫子仪	男	1989/6/11 0:00:00	深圳	8.jpg	编辑
9	李小菲	女	1989/6/1 0:00:00	沈阳	9.jpg	编辑

图 9-22　GridView 排序效果

9.3.4　GridView 的分页

GridView 控件提供了内置的分页功能，绑定数据后只要设置分页相关属性，即可自动完成分页功能，只须在分页导航按钮的单击事件

处理方法中添加代码，设置当前要显示的页索引并重新绑定数据即可。

【例 9-6】通过 GridView 控件实现数据记录分页显示功能。

(1) 在 WebSite9 网站中新建一个名为 GridViewBingding_3.aspx 的页面，在页面上添加一个 GridView 控件，并添加用于显示分页信息的 Label，页面设计如图 9-23 所示。

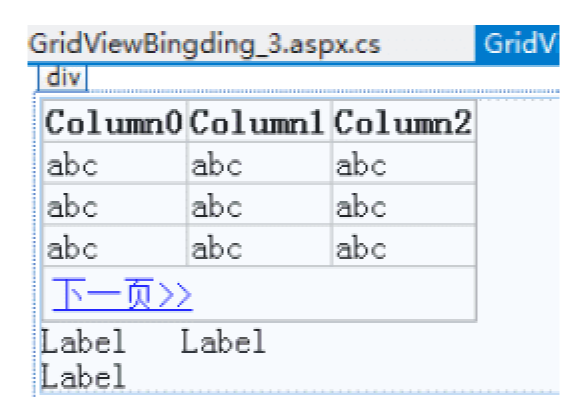


图 9-23　GridViewBingding_3.aspx 设计页面

(2) 页面代码如下。

```
<div>
        <asp:GridView ID="GridView1" runat="server" AllowPaging="True"
OnPageIndexChanging="GridView1_PageIndexChanging"
            PageSize="3" OnDataBound="GridView1_DataBound">
            <PagerSettings Mode="NextPrevious" NextPageText="下一页&gt;&gt;"
PreviousPageText="&lt;&lt;上一页" />
        </asp:GridView>
        <asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>
         
        <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label><br />
        <asp:Label ID="Label3" runat="server" Text="Label"></asp:Label><br />
</div>
```

(3) 在后台添加数据绑定的代码如下。

```
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
……
protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack) bindgrid();
    }
    void bindgrid()
    {
//查询 student 数据库获取结果集 ds
        string sqlconnstr =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;
        DataSet ds = new DataSet();
        using (SqlConnection sqlconn = new SqlConnection(sqlconnstr))
        {
          SqlDataAdapter sqld = new SqlDataAdapter("select * from student", sqlconn);
          sqld.Fill(ds, "tabstudent");
        }
//以数据集中名为 tabstudent 的 DataTable 作为数据源，为控件绑定数据
        GridView1.DataSource = ds.Tables["tabstudent"].DefaultView;
        GridView1.DataBind();    }
```

(4) 打开 GridView 数据属性设置窗口，为其设置分页相关属性，如图 9-24 所示。分页的设置主要有以下 3 个属性。

- AllowPaging：设置是否打开分页功能。
- PageIndex：当前显示的页索引。
- PageSize：设置每页包含的最大项数。

除了上述 3 种分页属性，还可展开 PageSettings 子项，在其中设置分页模式、分页按钮的显示文本等分页后的控件样式。其中 Mode 属性设置分页模式，共有 4 种可选模式，这里选择 NextPrevieus 模式。

(5) 设置完分页属性后就可以为分页导航按钮设置分页事件处理方法，如图 9-25 所示。

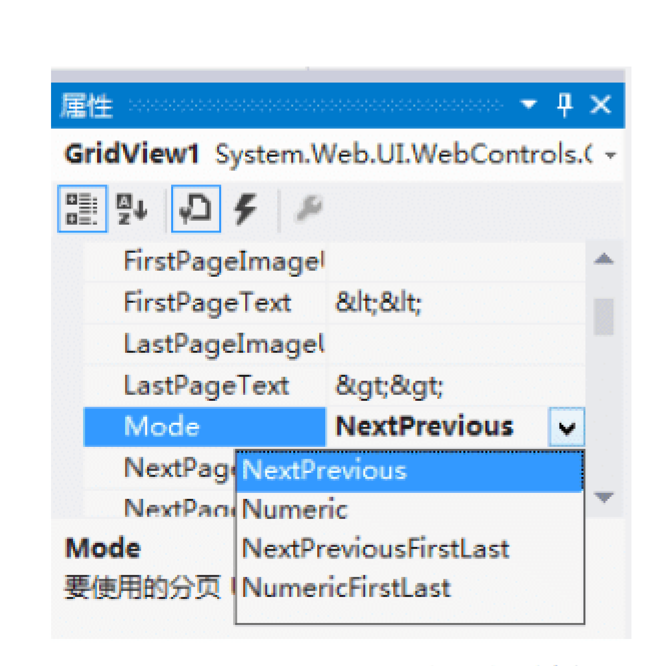



图 9-24　GridView 分页属性设置

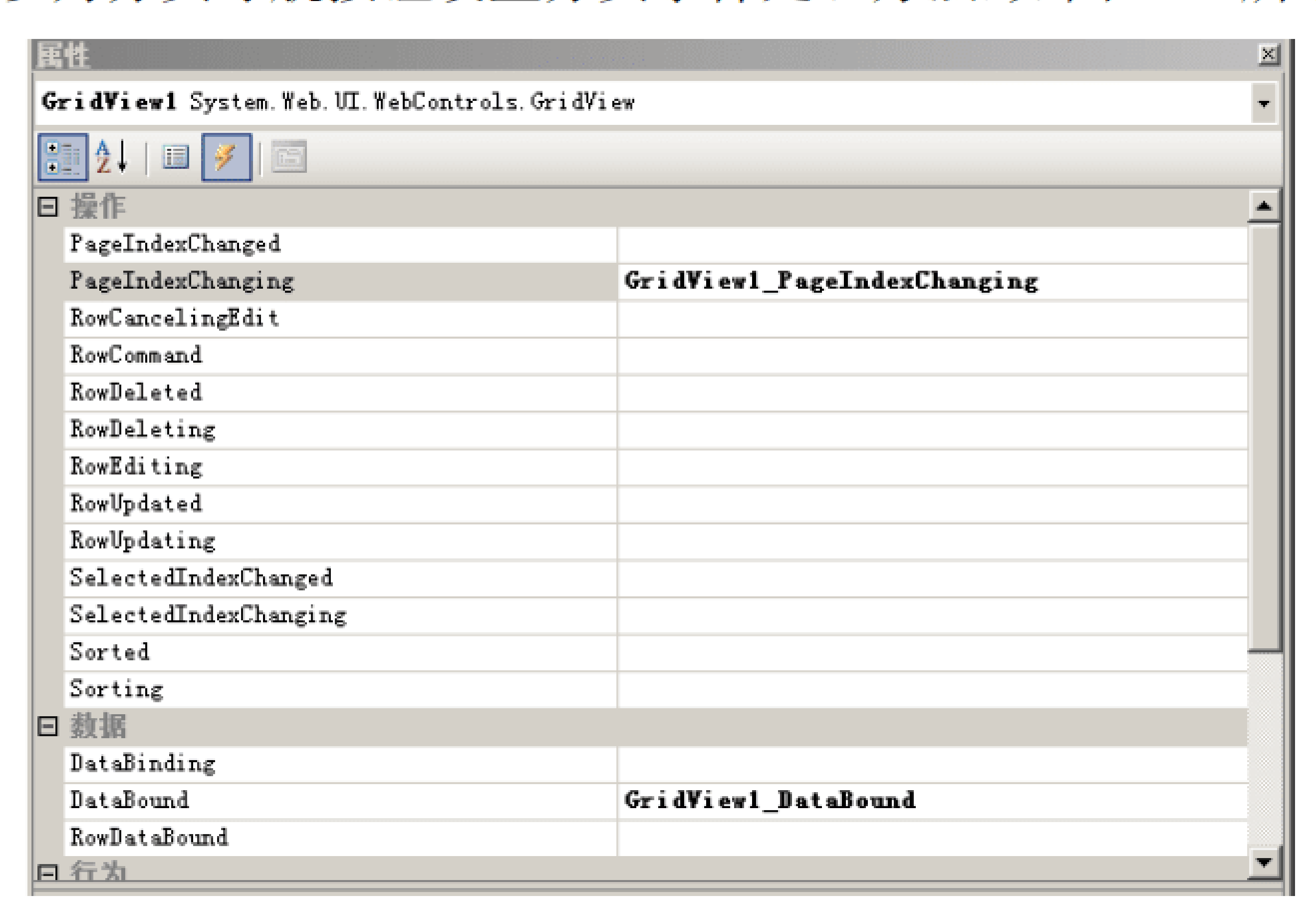


图 9-25　设置分页事件处理方法

图 9-25 中为 PageIndexChanging 事件设置了事件处理方法，该事件在分页导航按钮被单击时触发，并返回导航按钮所指示的，也就是控件中要显示的页的索引，在其事件处理方法中根据该索引设置要显示的页并重新绑定数据即可完成分页。另外还设置了 DataBound 事件的处理方法，用于在分页时重新绑定数据后设置 Label 控件显示分页信息，以及总共的页数、当前页数。

(6) 为事件处理方法添加代码如下。

```
protected void GridView1_PageIndexChanging(object sender, GridViewPageEventArgs e)
{
    //设置要显示的页的索引并重新绑定数据
    GridView1.PageIndex = e.NewPageIndex;
    bindgrid();
}
protected void GridView1_DataBound(object sender, EventArgs e)
{
    //分页数据绑定前设置当前页信息
    Label2.Text = "共" + (GridView1.PageCount).ToString() + "页";
    Label1.Text = "第" + (GridView1.PageIndex + 1) .ToString() + "页";
    Label3.Text = string.Format("总页数：{0}，当前页：{1}", GridView1.PageCount,
GridView1.PageIndex + 1) ;
}
```

(7) 运行效果如图 9-26 所示。



图 9-26　GridViewBingding_3.aspx 运行效果

9.4　DataList 和 FormView 控件

本节将介绍另外两种较为复杂的数据绑定控件：DataList 控件和 FormView 控件。与 GridView 一样，这两种服务器控件也用于呈现关系数据库集，但它们不像 GridView 控件那样以固定表格样式显示数据，而以自定义模板方式定制数据的呈现样式，这与 Gridview 的自定义模板列非常类似。DataList 和 FormView 控件以项为单位组织和呈现数据(GridView 以列为单位)，每一项对应于关系数据集的一条记录(行)。通过定义和设置不同的项模板定制每一项的显示样式，绑定数据后控件将按照项模板重复显示数据源的每条记录。呈现数据时 3 种控件对项的显示布局各不相同。DataList 控件提供两种页面布局：Table 和 Flow，在 Table 模式下在一个行列表中重复每个数据项，可以通过相关属性控制其按行显示或按列显示并设置行(列)中包含的最大项数；Flow 模式下在一行或者一列中重复显示数据项。FormView 控件默认每页显示一个数据项，通过分页导航访问每条记录。

在 DataList 和 FormView 控件中可以实现对关系数据集的编辑、更新、插入、删除和分页等数据处理功能。DataList 和 FormView 控件针对数据源控件提供内置的数据处理功能，只须某些配置即可自动完成，而针对其他类型的数据源公开特定的属性和事件通过编写代码来实现。

通过前面对 GridView 控件的详细介绍，可以看出，复杂数据绑定控件的用法不外以下 3 个方面：

- 数据的绑定与呈现；
- 数据的编辑、修改、添加、删除；
- 数据的分页和排序。

下面通过实例分别加以介绍，由于 3 种控件的用法类似，这里只针对 DataList 实现数据的呈现与绑定，针对 FormView 实现数据的编辑、增、删、改及分页。

另外还有一种基于项模板数据绑定控件：Repeater，它不提供任何内置的显示布局和内置的数据处理功能，只是按照定义好的项模板简单地重复显示数据。由于 Repeater 控件，通常只是用于以同一样式重复显示数据记录，而其项模板样式的定义和使用方法与 FormView 和 DataList 极为相似，因此不再提供实例，读者可以自行演练。

9.4.1　DataList 的数据绑定

DataList 控件中通过自定义模板设置数据的显示样式，它支持如下模板类型。

- ItemTemplate：包含一些 HTML 元素和控件，将为数据源中的每一行呈现一次这些 HTML 元素和控件。
- AlternatingItemTemplate：包含一些 HTML 元素和控件，将为数据源中的每两行呈现一次这些 HTML 元素和控件。通常，可以使用此模板来为交替行创建不同的外观。例如，指定一个与在 ItemTemplate 属性中指定的颜色不同的背景色。
- SelectedItemTemplate：包含一些元素，当用户选择 DataList 控件中的某一项时将呈现这些元素。通常，可以使用此模板来通过不同的背景色或字体颜色直观地区分选定的行。还可以通过显示数据源中的其他字段来展开该项。
- EditItemTemplate：指定当某项处于编辑模式中时的布局。此模板通常包含一些编辑控件，如 TextBox 控件。
- HeaderTemplate 和 FooterTemplate：包含在列表的开始和结束处分别呈现的文本和控件。
- SeparatorTemplate：包含在每项之间呈现的元素。典型的示例可能是一条直线(使用 HR 元素)。

设计者需要根据不同的需要定义不同类型的项模板，DataList 控件根据项的运行时状态自动加载相应的模板显示数据。例如，当某一项被选定后将会以 SelectedItemTemplate 模板呈现数据，编辑功能被激活时将以 EditItemTemplate 模板呈现数据。

下面的【例 9-7】介绍了如何通过设置模板为 DataList 控件定义数据的呈现样式并完成数据绑定。

【例 9-7】为 DataList 控件实现数据绑定，以按要求显示数据记录。

(1) 在 WebSite9 网站中新建一个名为 DataListBingding.aspx 的页面，在页面上添加一个 DataList 控件。

(2) 为 DataList 编辑、设置项模板并进行显示字段映射。

在 VS 环境中使用 DataList 控件的快捷任务面板进入模板的编辑页面，如图 9-27 所示。

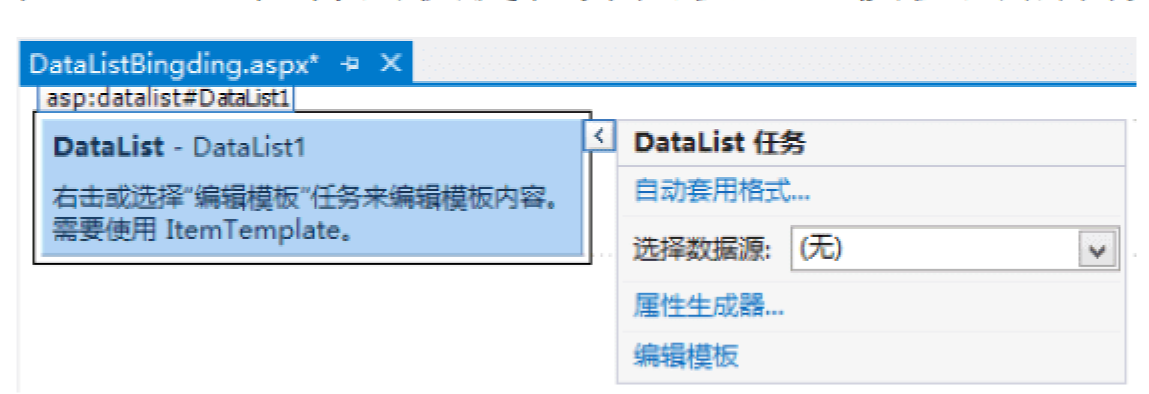


图 9-27　打开 DataList 的模板编辑器

单击“编辑模板”按钮后进入模板编辑界面，如图 9-28 所示。

在这里只实现 DataList 控件的数据绑定，所以只简单地定义一个 ItemTemplate，单击模板类型后编辑 ItemTemplate 模板样式如图 9-29 所示。

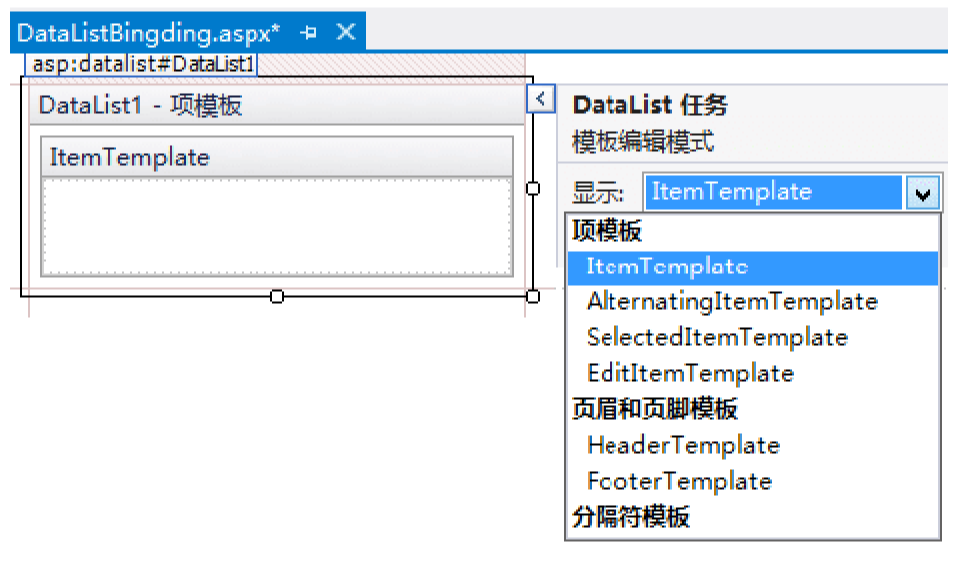



图 9-28　模板编辑界面

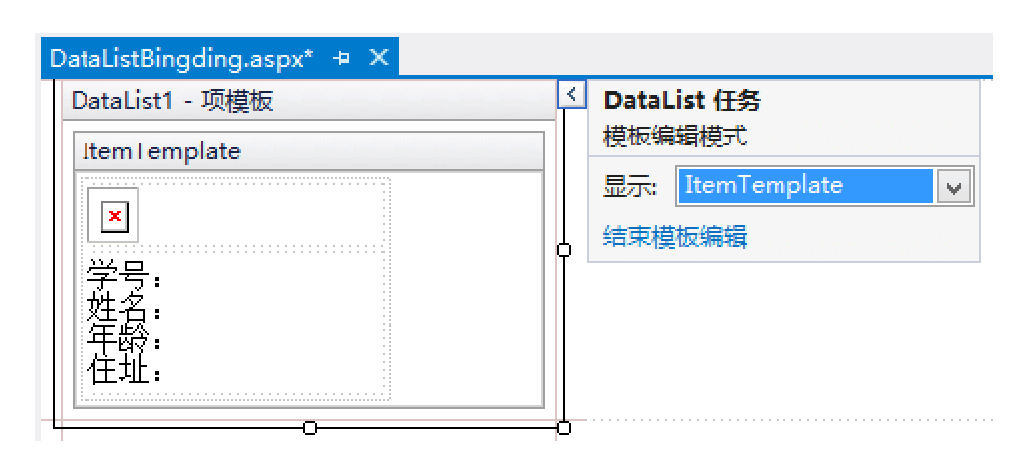


图 9-29　ItemTemplate 模板样式

ItemTemplate 模板样式中，包含一个 2 行 1 列的 HTML Table，第 1 行显示图片，第 2 行显示记录中的其他字段。回顾 GridView 控件中，设置绑定列时需要同时设置绑定列到数据字段之间的数据映射，DataList 控件中的项模板显示数据源每条记录中的各个字段，也需要将模板中的显示控件映射到相应字段，才能在数据绑定后在模板项中显示正确的数据。数据映射通过绑定表达式完成，在项模板中各个显示控件的页面代码中添加如下绑定表达式：<%# Eval("XXX") %>，其中 Eval 方法用于读取数据绑定后当前显示项中所呈现的数据项(某条记录)的相应字段数据，Eval 方法的参数 XXX 指定记录中要显示的字段名。可以这样来理解<%# Eval("XXX") %>表达式，当在后台代码中为某种数据绑定控件(如这里的 DataList)设置数据源并进行数据绑定后，运行时数据源中的记录就会自动与显示项关联，有这种关联作为上下文，只要指定字段名就可以访问到该记录中的字段数据。因为 Eval 方法需要在数据上下文中读取数据，因此<%# Eval("XXX") %>表达式只能用在数据绑定控件的模板定义中。

定义模板后的页面代码如下。

```
<div>
    <asp:DataList ID="DataList1" runat="server" Height="354px" RepeatColumns="3"
        HorizontalAlign="Justify" RepeatDirection="Horizontal">
        <ItemTemplate>
            <table style="width: 154px; height: 111px">
                <tr>
                    <td style="width: 100px">
                        <img alt="照片" src='./images/<%# Eval("Photo") %>' /></td>
                </tr>
                <tr>
                    <td style="width: 100px">
                        学号：<%# Eval("No") %><br />
                        姓名：<%# Eval("Name") %><br />
                        年龄：<%# Eval("birthday") %><br />
                        住址：<%# Eval("Address") %></td>
                </tr>
            </table>
        </ItemTemplate>
    </asp:DataList>
</div>
```

项模板第 1 行图片控件中的表达式<%# Eval("photo") %>读取数据源记录中的 photo 字

段值作为图片名称。

(3) 设置 DataList 的布局属性，采用 Table 布局，每行显示 5 个项，按行显示，如图 9-30 所示。

属性
DataList1 System.Web.UI.WebControls.D

Height	354px
HorizontalAlign	Justify
RepeatColumns	5
RepeatDirection	Horizontal
RepeatLayout	Table
Width	

行为
RepeatColumns
要用于该布局的列的数目。

图 9-30　设置 DataList 布局属性

(4) 在页面后台类中添加数据绑定代码如下。

```
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
……
protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack) listbind();
    }
    void listbind()
    {
        string sqlconnstr =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString; ;
        DataSet ds = new DataSet();
        using (SqlConnection sqlconn = new SqlConnection(sqlconnstr))
        {
            SqlDataAdapter sqld = new SqlDataAdapter("select * from student", sqlconn);
            sqld.Fill(ds, "tabstudent");
        }
        //以数据集中名为 tabstudent 的 DataTable 作为数据源，为控件绑定数据
        DataList1.DataSource = ds.Tables["tabstudent"].DefaultView;
        DataList1.DataBind();
    }
```

(5) 页面的运行效果如图 9-31 所示。



图 9-31　DataListBinding.aspx 页面运行效果

9.4.2　FormView 控件的数据呈现和处理

FormView 控件提供了内置的数据处理功能，只须绑定到支持这些功能的数据源控件，并进行配置，无须编写代码就可以实现对数据的分页和增删改功能。要使用 FormView 内置的增删改功能，需要为更新操作提供 EditItemTemplate 和 InsertItemTemplate 模板，FormView 控件显示指定的模板以提供修改记录内容的用户界面。每个模板都包含用户可

以单击以执行编辑或插入操作的命令按钮。用户单击命令按钮时，FormView 控件使用指定的编辑或插入模板重新显示绑定记录以允许用户修改记录。插入或编辑模板通常包括一个允许用户显示空白记录的【插入】按钮或保存更改的【更新】按钮。用户单击【插入】或【更新】按钮时，FormView 控件将绑定值和主键信息传递给关联的数据源控件，该控件执行相应的更新。例如，SqlDataSource 控件使用更改后的数据作为参数值来执行 SQL Update 语句。

由于 FormView 控件的各个项通过自定义模板来呈现，因此控件并不提供内置的实现某一功能(如删除)的特殊按钮类型，而是通过按钮控件的 CommandName 属性与内置的命令相关联。FormView 提供如下命令类型(区分大小写)。

- Edit：引发此命令控件转换到编辑模式，并用已定义的 EditItemTemplate 呈现数据。
- New：引发此命令控件转换到插入模式，并用已定义的 InsertItemTemplate 呈现数据。
- Update：此命令将使用用户在 EditItemTemplate 界面中的输入值在数据源中更新当前所显示的记录，引发 ItemUpdating 和 ItemUpdated 事件。
- Insert：此命令用于将用户在 InsertItemTemplate 界面中的输入值在数据源中插入一条新的记录。引发 ItemInserting 和 ItemInserted 事件。
- Delete：此命令删除当前显示的记录。引发 ItemDeleting 和 ItemDeleted 事件。
- Cancel：在更新或插入操作中取消操作和放弃用户输入值，然后控件会自动转换到 DefaultMode 属性指定的模式。

在命令所引发的事件中，可以执行一些额外的操作。例如，对于 Update 和 Insert 命令，因为 ItemUpdating 和 ItemInserting 事件是在更新或插入数据源之前触发的，可以在 ItemUpdating 和 ItemInserting 事件中先判断用户的输入值进行验证，满足要求后才访问数据库，否则取消操作。

下面通过【例 9-8】演示如何使用 FormView 控件完成数据的分页呈现，编辑、更新、删除、添加等数据处理功能。

【例 9-8】 在 FormView 控件中实现数据记录的分页呈现、编辑、更新、删除和添加操作。

(1) 在 WebSite9 网站中新建一个名为 FormViewBinding.aspx 的页面，在页面上添加 1 个 FormView 控件。

(2) 为 FormView 控件添加并编辑项模板，由于要实现数据的更新和插入操作，需要 3 种项模板 ItemTemplate、EditItemTemplate 和 InsertItemTemplate，分别在显示、更新和插入状态下呈现数据。在 FormView 控件中也提供模板编辑界面(如 DataList 控件)，这里直接在页面代码中进行编辑，首先编辑 EditItemTemplate 页面代码如下。

```
    <EditItemTemplate>
        <table style="width:100%;">
            <tr>
                <td width="40%"><asp:Label ID="Label1" runat="server" Text="学号"
Width="100%"></asp:Label> </td>
```

```
                    <td width="60%"><asp:Label ID="noLabel1" runat="server" Text='<%# Eval("No")
%>' /></td>
                </tr>
                <tr>
                    <td width="40%"><asp:Label ID="Label2" runat="server" Text="姓名"
Width="100%"></asp:Label> </td>
                    <td width="60%"><asp:TextBox ID="nameTextBox" runat="server" Text='<%#
Bind("Name") %>' /></td>
                </tr>
                <tr>
                    <td width="40%"><asp:Label ID="Label3" runat="server" Text="出生日期"
Width="100%"></asp:Label> </td>
                    <td width="60%"> <asp:TextBox ID="birthTextBox" runat="server" Text='<%#
Bind("birthday") %>' /></td>
                 </tr>
                  <tr>
                      <td width="40%"><asp:Label ID="Label4" runat="server" Text="地址"
Width="100%"></asp:Label> </td>
                      <td width="60%"><asp:TextBox ID="addressTextBox" runat="server"
Text='<%# Bind("Address") %>' /></td>
                  </tr>
                  <tr>
                      <td width="40%"> </td>
                      <td width="60%" align="center">
                      <asp:LinkButton ID="UpdateButton" runat="server" CausesValidation="True"
CommandName="Update" Text="更新" />
                      <asp:LinkButton ID="UpdateCancelButton" runat="server"
CausesValidation="False" CommandName="Cancel" Text="取消" />
                        </td>
                   </tr>
               </table>
     </EditItemTemplate>
```

编辑状态模板中用 1 个 Label 控件和 1 个 TextBox 控件代表数据源中的一个字段，其中 TextBox 控件为绑定字段，共 4 行，对应于数据源记录中的 4 个字段。与 DataList 中不同，这里 TextBox 控件的绑定表达式为<%# Bind("Address") %>，表达式标记中调用了 Bind 方法(DataList 中使用 Eval 方法)，Bind 方法构成与数据源的双向映射，通过双向映射配合数据源控件可以完成控件内置的更新操作(仅对数据源控件有效)。而 DataList 控件中使用的 Eval 方法为单向映射不能更新数据。

用于数据显示的 ItemTemplate 和用于插入的 InsertItemTemplate 与之类似，这里不再赘述。

(3) 配置完成各个项模板后，为 FormView 配置分页，由于分页功能是内置的，只需要设置 FormView 控件的分页属性，如图 9-32 所示。

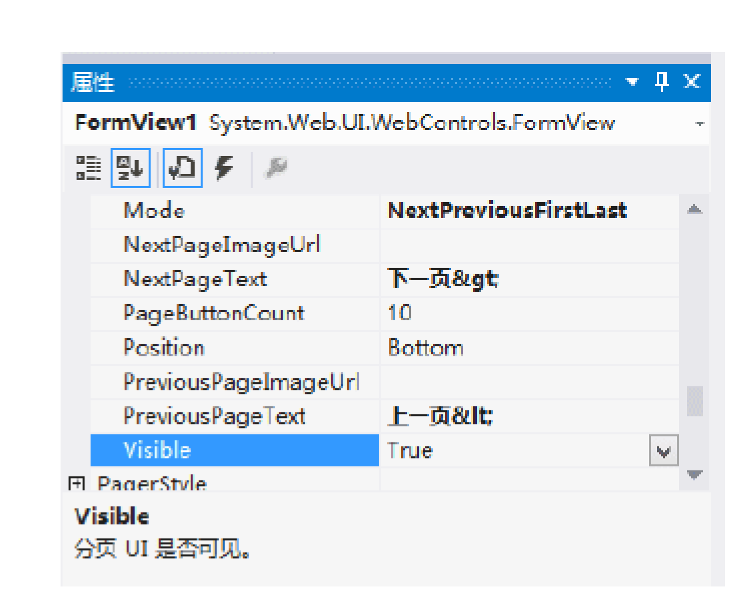


图 9-32　为 FormView 设置分页属性

(4) 调整 FormView 控件的外观，设置页眉模板 HeaderTemplate 完成整个页面的设计，完整的

FormView 控件设计代码如下。

```
    <asp:FormView ID="FormView1" runat="server" DataKeyNames="no"
    DataSourceID="SqlDataSource1" AllowPaging="True" CellPadding="4"    ForeColor="#333333"
Width="231px">
    <PagerSettings Mode="NextPreviousFirstLast" NextPageText="下一页&amp;gt;"
PreviousPageText="上一页&amp;lt;" />
    <FooterStyle BackColor="#990000" Font-Bold="True" ForeColor="White" />
    <RowStyle BackColor="#FFFBD6" ForeColor="#333333" />
    <EditItemTemplate>
      <table style="width:100%;">
        <tr>
          <td width="40%"><asp:Label ID="Label1" runat="server" Text="学号"
Width="100%"></asp:Label> </td>
          <td width="60%"><asp:Label ID="noLabel1" runat="server" Text='<%# Eval("No") %>'
/></td>
       </tr>
       <tr>
          <td width="40%"><asp:Label ID="Label2" runat="server" Text="姓名"
Width="100%"></asp:Label> </td>
          <td width="60%"><asp:TextBox ID="nameTextBox" runat="server" Text='<%# Bind("Name")
%>' /></td>
       </tr>
       <tr>
           <td width="40%"><asp:Label ID="Label3" runat="server" Text="出生日期"
Width="100%"></asp:Label> </td>
          <td width="60%"> <asp:TextBox ID="birthTextBox" runat="server" Text='<%#
Bind("birthday") %>' /></td>
     </tr>
       <tr>
          <td width="40%"><asp:Label ID="Label4" runat="server" Text="地址"
Width="100%"></asp:Label> </td>
          <td width="60%"><asp:TextBox ID="addressTextBox" runat="server" Text='<%#
Bind("Address") %>' /></td>
       </tr>
       <tr>
           <td width="40%"> </td>
           <td width="60%" align="center">
             <asp:LinkButton ID="UpdateButton" runat="server" CausesValidation="True"
CommandName="Update" Text="更新" />
            <asp:LinkButton ID="UpdateCancelButton" runat="server" CausesValidation="False"
CommandName="Cancel" Text="取消" />
           </td>
      </tr>
        </table>
         </EditItemTemplate>
         <InsertItemTemplate>
           <table style="width:100%;">
             <tr>
               <td width="40%"><asp:Label ID="Label2" runat="server" Text="姓名"
Width="100%"></asp:Label> </td>
               <td width="60%"><asp:TextBox ID="nameTextBox" runat="server" Text='<%#
```

```
Bind("Name") %>' /></td>
                        </tr>
                        <tr>
                              <td width="40%"><asp:Label ID="Label3" runat="server" Text="出生日期"
Width="100%"></asp:Label> </td>
                              <td width="60%"><asp:TextBox ID="birthTextBox" runat="server" Text='<%#
Bind("birthday") %>' /></td>
                      </tr>
                      <tr>
                               <td width="40%"><asp:Label ID="Label4" runat="server" Text="地址"
Width="100%"></asp:Label> </td>
                               <td width="60%"><asp:TextBox ID="addressTextBox" runat="server"
       Text='<%# Bind("Address") %>' /></td>
                    </tr>
                    <tr>
                               <td width="40%"> </td>
                               <td width="60%" align="center">
                                   <asp:LinkButton ID="InsertButton" runat="server" CausesValidation="True"
CommandName="Insert" Text="插入" />
                                   <asp:LinkButton ID="InsertCancelButton" runat="server" CausesValidation="False"
CommandName="Cancel" Text="取消" />
                               </td>
                    </tr>
                          </table>
                          </InsertItemTemplate>
                          <ItemTemplate>
                              <table style="width:100%;">
                                   <tr>
                                      <td width="40%"><asp:Label ID="Label1" runat="server" Text="学号"
Width="100%"></asp:Label> </td>
                                      <td width="60%"><asp:Label ID="noLabel" runat="server" Text='<%#
Eval("No") %>' /></td>
                                  </tr>
                                  <tr>
                                      <td width="40%"><asp:Label ID="Label2" runat="server" Text="姓名"
Width="100%"></asp:Label> </td>
                                      <td width="60%"><asp:Label ID="nameLabel" runat="server" Text='<%#
Bind("Name") %>' /></td>
                                  </tr>
                                  <tr>
                                      <td width="40%"><asp:Label ID="Label3" runat="server" Text="出生日期"
Width="100%"></asp:Label> </td>
                                      <td width="60%"><asp:Label ID="birthLabel" runat="server" Text='<%#
Bind("birthday") %>' /></td>
                                  </tr>
                                  <tr>
                                       <td width="40%"><asp:Label ID="Label4" runat="server" Text="地址"
Width="100%"></asp:Label> </td>
                                      <td width="60%"><asp:Label ID="addressLabel" runat="server"
       Text='<%# Bind("Address") %>' /></td>
                               </tr>
                               <tr>
```

```
                        <td width="40%"> </td>
                        <td width="60%" align="right">
                        <asp:LinkButton ID="NewButton" runat="server" CausesValidation="True"
CommandName="New" Text="新建" />
                        <asp:LinkButton ID="EditButton" runat="server" CausesValidation="False"
CommandName="Edit" Text="更新" />
                        <asp:LinkButton ID="DeleteButton" runat="server" CausesValidation="False"
CommandName="Delete" Text="删除" />
                        </td>
                    </tr>
                </table>
            </ItemTemplate>
    <PagerStyle BackColor="#FFCC66" ForeColor="#333333" HorizontalAlign="Center" />
    <HeaderStyle BackColor="#990000" Font-Bold="True" ForeColor="White" />
        <HeaderTemplate>
            学生详细信息
        </HeaderTemplate>
    </asp:FormView>
```

页面设计外观如图 9-33 所示。

页面设计完成后为控件创建并配置数据源控件，需要在数据源控件中配置用于实现增删改查功能的查询语句以支持 FormView 控件中的相应命令按钮。

(5) 在页面上添加一个 SqlDataSource 数据源控件，过程如下。

首先为数据源控件配置数据库连接，依然使用前面介绍过的 mydatabase.mdf，如图 9-34 所示。

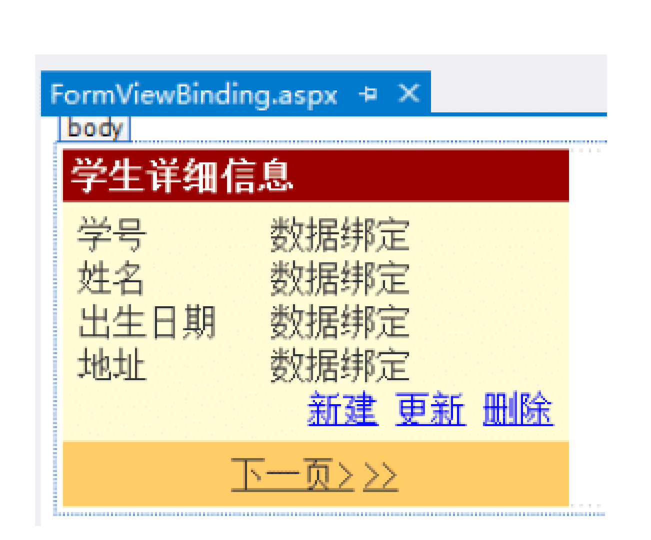


图 9-33　FormViewBinding 页面设计效果

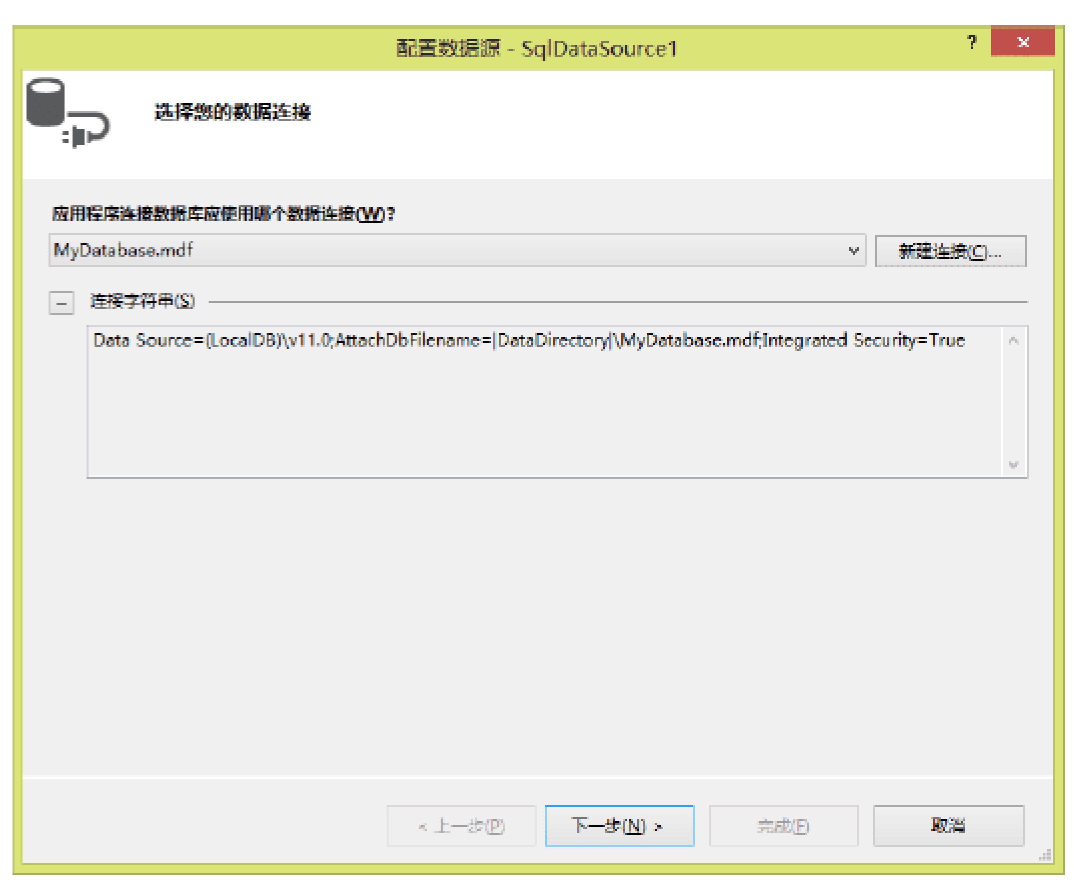


图 9-34　为数据源控件配置连接

单击下一步配置连接字符串，系统将自动生成 mydatabose.mdf 的连接字符串并保存，如果已经按照前面给出的手动方式将连接字符串添加到 web.config 文件中，则可以跳过此步。

然后为数据源配置查询如图 9-35 所示。

完成后，为 SqlDataSource 数据源控件配置更新、插入和删除语句。

首先，打开数据源控件的【属性】窗口在其数据属性部分设置 DeleteCommandType

属性，该属性支持两种枚举值，Text 和 StoredProcedure，前者使用 Sql 语句实现删除操作，后者用数据库中的存储过程实现，例中设置为 Text，如图 9-36 所示。

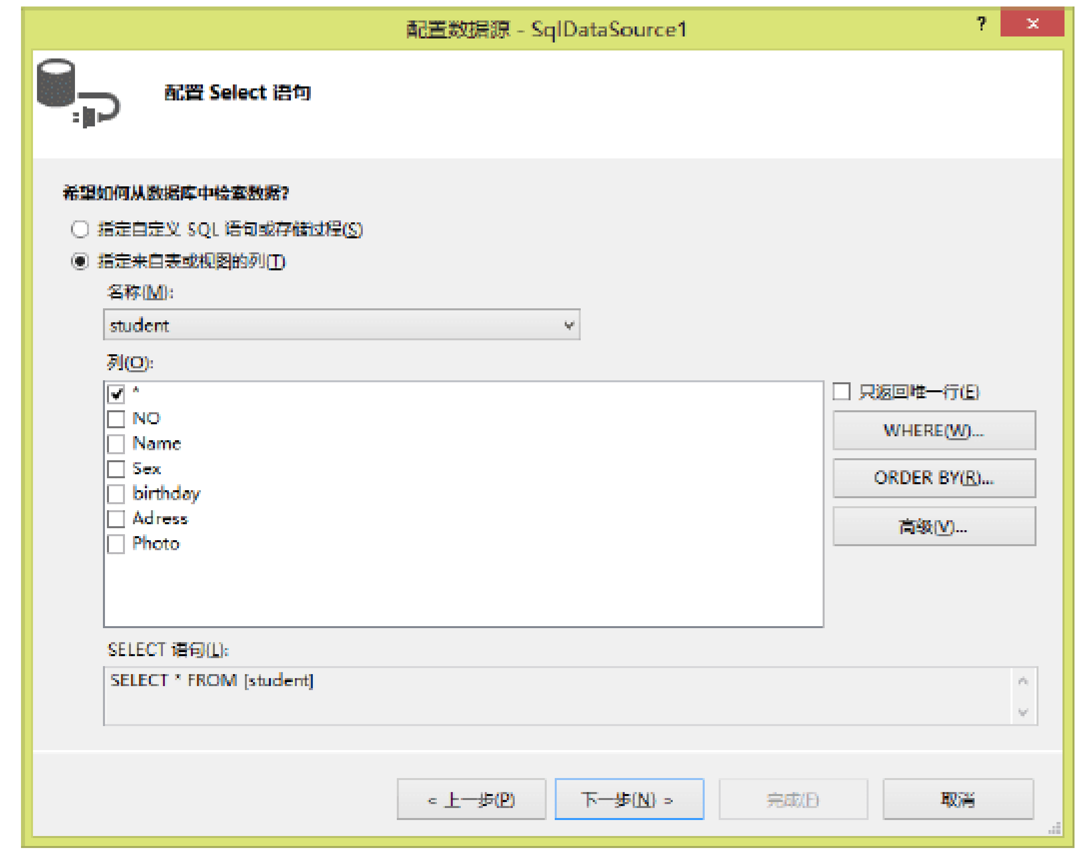




图 9-35　配置查询

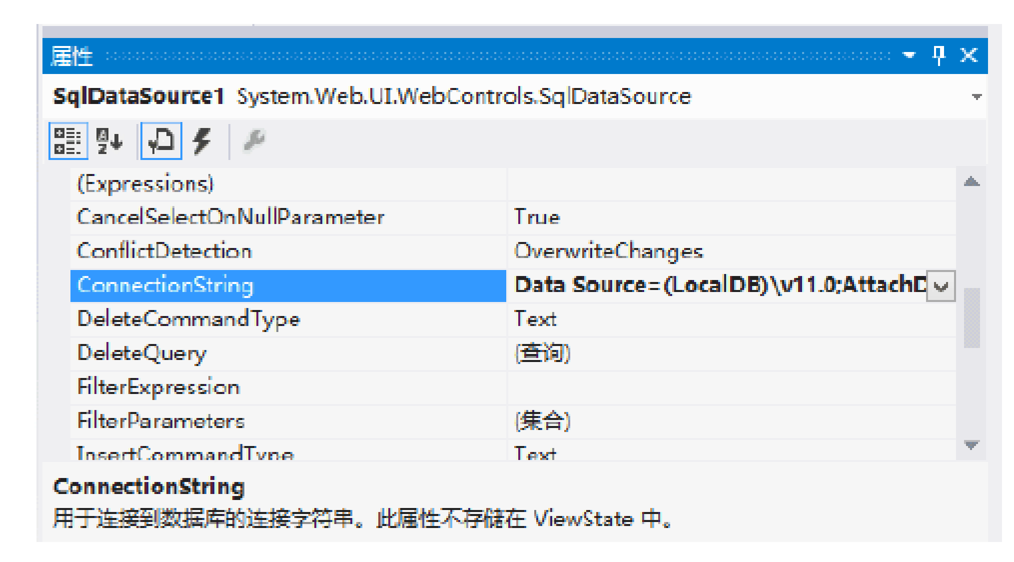


图 9-36　为 SqlDataSource 控件配置属性

然后为查询操作添加 Sql 语句完成删除操作，通过 DeleteQuery 属性打开删除命令的【命令和参数编辑器】对话框，如图 9-37 所示。

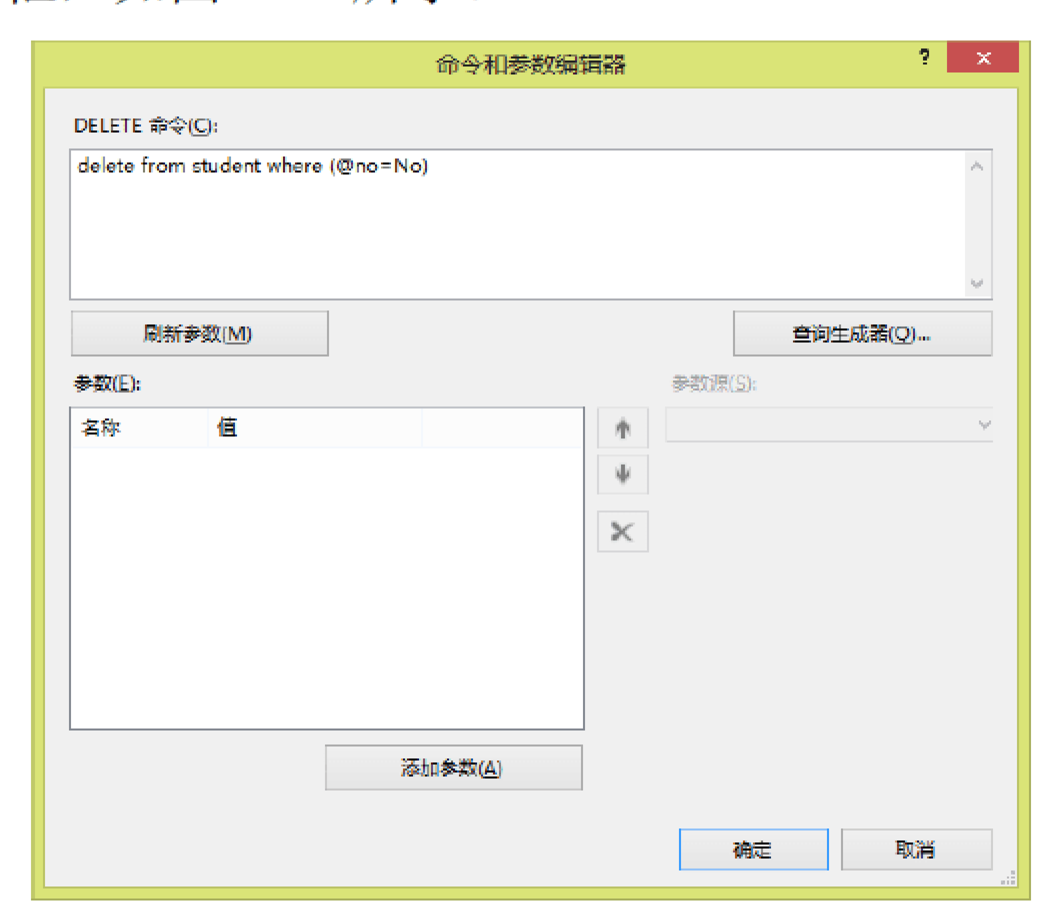


图 9-37　【命令和参数编辑器】对话框

在 DELETE 命令一栏输入删除语句：DELETE FROM student WHERE (No = @No)。这里需要注意 Delete 语句的参数 No = @No，正因为在编辑模板中为数据显示项和数据源记录字段之间建立了双向绑定，才可以直接将绑定字段名 no 作为参数名完成删除语句，系统将自动获取编辑模板中的相应显示项的当前显示值作(对于插入和更新操作是输入值)为参数执行 Sql 语句。

用同样的方法为更新操作添加如下 Update 语句。

```
UPDATE student SET Name = @Name,birthday=@birthday,Address=@Address where No=@No
```

为插入操作添加如下 Insert 语句。

```
INSERT INTO student(Name, birthday, Address) VALUES (@Name,@birthday,@Address)
```

上述设置将会在页面前台生成如下 SqlDataSource 控件定义代码。

```
    <asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="Data
Source=(LocalDB)\v11.0;AttachDbFilename=|DataDirectory|\MyDatabase.mdf;Integrated Security=True"
DeleteCommand="DELETE FROM student WHERE (NO = @no)" InsertCommand="INSERT INTO
student(Name, birthday, Address) VALUES (@Name,@birthday,@Address)"
ProviderName="System.Data.SqlClient" SelectCommand="SELECT * FROM [student]"
UpdateCommand="UPDATE student SET Name = @Name,birthday=@birthday,Address=@Address
where No=@No">
                <DeleteParameters>
                    <asp:Parameter DefaultValue="NO" Name="No" />
                </DeleteParameters>
                <InsertParameters>
                    <asp:Parameter Name="Name" />
                    <asp:Parameter Name="birthday" />
                    <asp:Parameter Name="Address" />
                </InsertParameters>
                <UpdateParameters>
                    <asp:Parameter Name="No" />
                    <asp:Parameter Name="Name" />
                    <asp:Parameter Name="birthday" />
                    <asp:Parameter Name="Address" />
                </UpdateParameters>
            </asp:SqlDataSource>
```

(6) 到此为止，已全部完成 FormView 控件的设计，可以按 F5 键运行效果，如图 9-38 和图 9-39 所示。

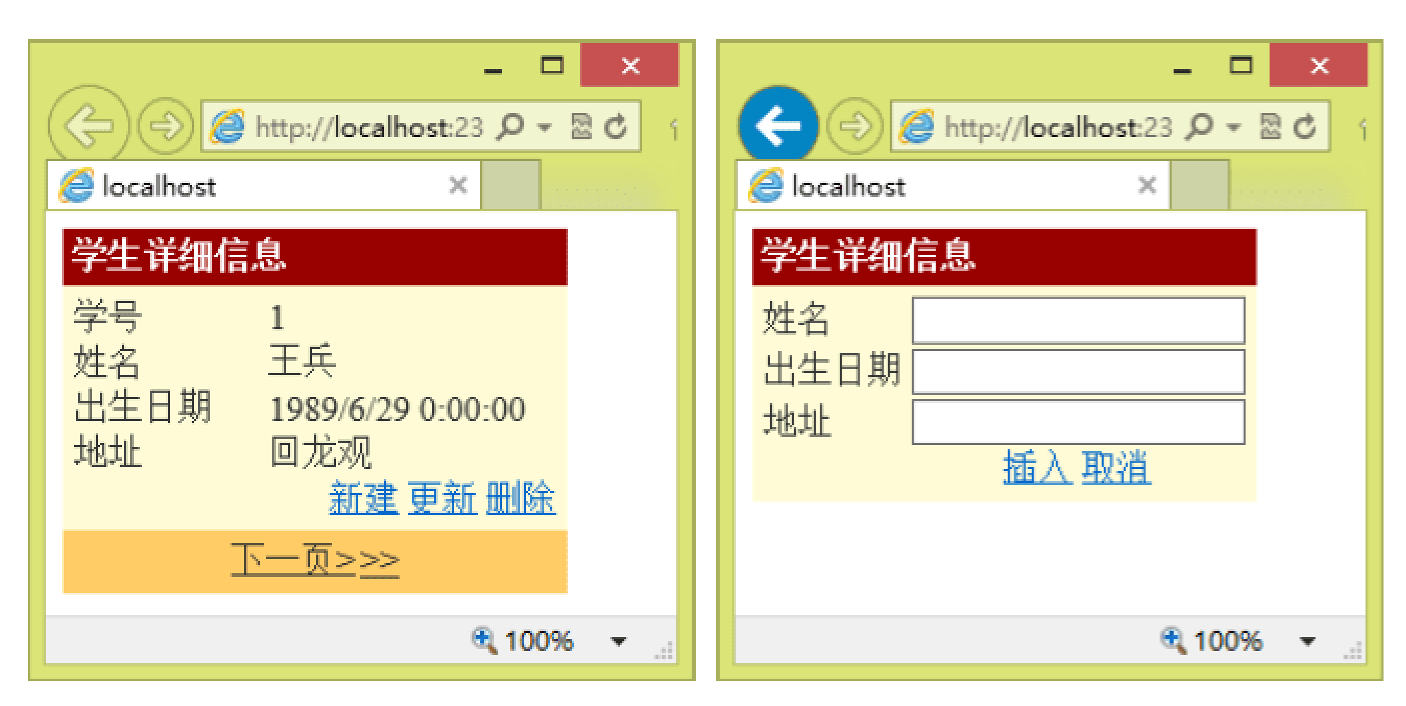


图 9-38　FormViewBinding.aspx 的显示和插入界面

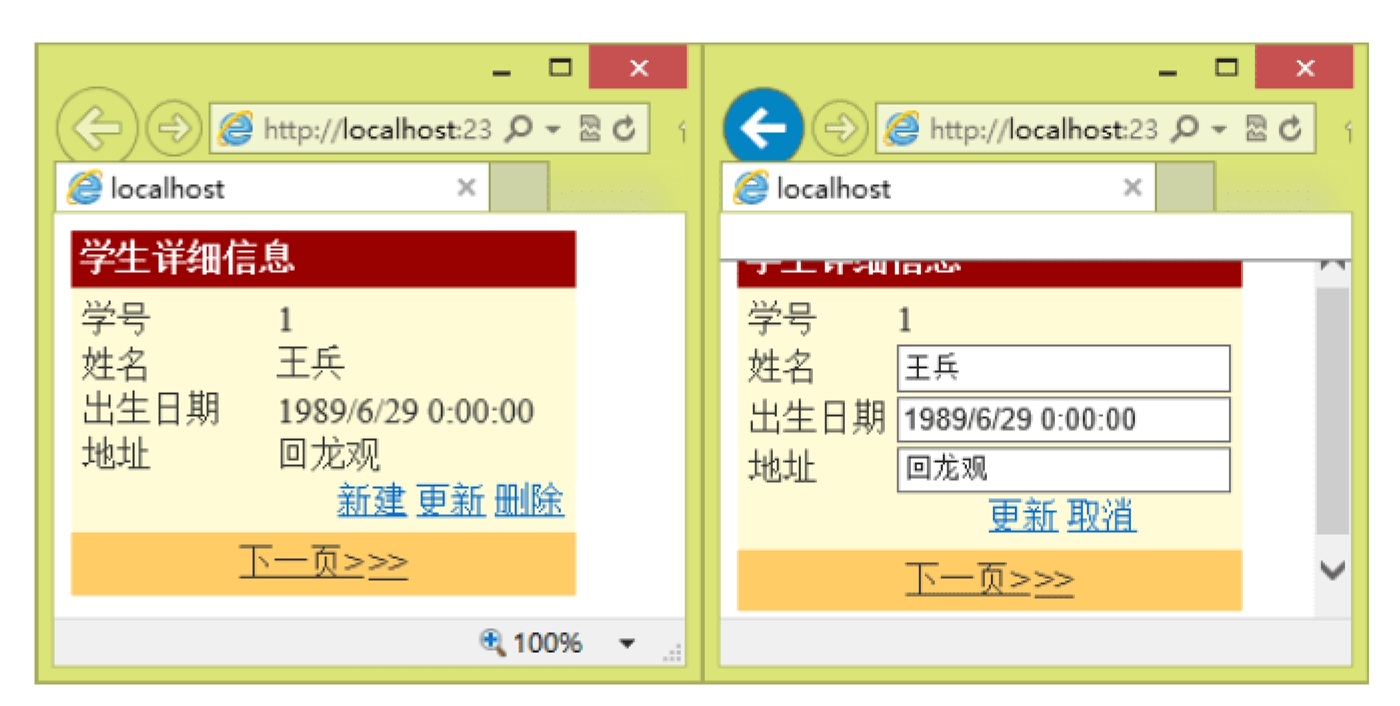


图 9-39　FormViewBinding.aspx 的显示和更新界面

9.5　本章小结

本章全面讲述了 ASP.NET 中的数据绑定技术。首先介绍了数据绑定的概念，然后介绍单值和列表控件的数据绑定操作，接着介绍了 ASP.NET 中使用得最多的数据绑定控件 GridView，最后介绍 DataList 和 FromView 控件的数据绑定操作。通过本章的学习，读者应该能够掌握将控件与数据查询结果进行绑定的操作技术，更方便地对数据库数据进行增、删、改、查及呈现操作。

9.6　思考和练习

1. 如何在.aspx 页面中为控件的 Text 属性绑定单列值？
2. 常用的可以绑定数据的列表控件有哪些？
3. 添加一个网页，利用 GridView 实现数据的分页显示。
4. 添加一个网页，利用 DataList 实现数据的绑定显示。
5. 添加一个网页，利用 FormView 控件实现数据的插入、修改和删除操作，FormView 界面及布局自定义。

第10章　ASP.NET AJAX

ASP.NET AJAX 采用异步编程方式，最大的特点是提供对客户端脚本的自动管理。利用 ASP.NET AJAX 服务器控件，程序员可以使用 ASP.NET 中现有的各种控件实现局部页面更新。

本章首先介绍了 ASP.NET AJAX 的基础知识，然后详细介绍 ASP.NET AJAX 的主要服务器控件的用法，最后介绍 ASP.NET AJAX Control Extenders 扩展控件。

本章的学习目标：

- 掌握 ASP.NET AJAX 的基本知识。
- 掌握 ASP.NET AJAX 主要控件的用法。
- 了解 ASP.NET AJAX Control Extenders 扩展控件。

10.1　ASP.NET AJAX 概述

Ajax(Asynchronous JavaScript and XML)技术是由 Jesse James Garrett 提出的，是综合异步通信、JavaScript 以及 XML 等多种网络技术新的编程方式。从用户看到的实际效果来看，也可以形象地称之为无页面刷新技术。

2005 年，Microsoft 公司在专业开发人员大会上宣布将在 ASP.NET 上实现 Ajax 功能(开发代号为 Atlas)，主要是为了充分利用客户端 JavaScript、DHTML 和 XMLHttpRequest 对象。目的是帮助开发人员创建更具交互性的支持 Ajax 的 Web 应用程序。直到 2007 年 1 月，Microsoft 公司才真正推出了具有 Ajax 风格的方便的异步编程模型，这就是 ASP.NET AJAX 1.0。同时为了与其他的 Ajax 技术区分，Microsoft 公司用大写的 AJAX，并在其前面加上 ASP.NET。

Ajax 技术的主要内容包括：基于 Web 标准 XHTML+CSS 的表示；使用 DOM(Document Object Model)进行动态显示及交互；使用 XML 和 XSLT 进行数据交换及相关操作；使用 XMLHttpRequest 进行异步数据检索；使用 JavaScript 将所有的东西绑定在一起等。

Ajax 技术的最大优点就是能在不更新整个页面的前提下维护数据。这使得 Web 应用程序更为迅捷地回应用户动作，并避免了在网络上发送那些没有改变过的信息。

ASP.NET AJAX 1.0 是以可以在 ASP.NET 2.0 之上安装的单独一个下载的形式发布的。从. NET Framework 3.5 开始，所有这些特性都成为 ASP.NET 所固有的，这意味着在构建或部署应用时，不再需要下载和安装单独的 ASP.NET AJAX 安装文件。

当在 VS 中创建针对.NET Framework 的 ASP.NET 网站时，VS 会自动在 web.config 文

件里添加适当的 AJAX 注册设置。另外，核心 ASP.NET AJAX 服务器控件会出现在工具箱中。

对于 Web 开发来说，ASP.NET AJAX 从基础框架实现，到客户端与服务器的通信，都发生了翻天覆地的变化。相对于 ASP.NET 来说，ASP.NET AJAX 是一种更为成熟的 Web 开发技术。

下面介绍 ASP.NET AJAX 主要控件 ScriptManager、UpdatePanel、UpdateProgress 和 Timer 的用法。

10.2　ScriptManager 控件

ScriptManager 控件是 ASP.NET AJAX 的核心，它提供处理页面上的所有 ASP.NET AJAX 控件(UpdatePanel、UpdateProgress 等)的支持，没有该控件的存在其他 ASP.NET AJAX 控件是不能工作的，并且所有需要支持 ASP.NET AJAX 的 ASP.NET 页面上只能有一个 ScriptManager 控件。另外，ScriptManager 控件还可以生成相关的客户端代理脚本，以便能够在客户端脚本中访问 Web Service。

10.2.1　ScriptManager 控件的属性和方法

要使用 ASP.NET AJAX 提供的功能，必须在网页中包含一个 ScriptManager 控件。如果在网站中添加一个【AJAX Web 窗体】网页，ASP.NET 会自动在该网页中添加一个 ScriptManager 控件标记，代码如下。

```
<asp:ScriptManager ID="ScriptManager1" runat="server" />
```

但是，如果在网站中添加一个【Web 窗体】网页，ASP.NET 就不会自动添加这个控件，此时需要从【工具箱】中将其拖动到页面中，使其生成如下代码。

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
```

ScriptManager 控件提供了很多属性和方法，用于对客户端脚本进行各种复杂的管理。如表 10-1 所示列出了其常用属性和方法。

表 10-1　ScriptManager 控件的常用属性和方法

属性/方法	说　明
AsyncPostBackErrorMessage 属性	异步回传发生错误时的自定义提示错误信息
AsyncPostBackTimeout 属性	异步回传时超时限制，默认值为 90，单位为秒
EnablePartialRendering 属性	是否支持页面的局部更新，默认值为 True，一般不需要修改
ScriptMode 属性	指定 ScriptManager 发送到客户端的脚本的模式，有 4 种模式：Auto、Inherit、Debug、Release，默认值为 Auto

(续表)

属性/方法	说　明
ScriptPath 属性	设置所有的脚本块的根目录，作为全局属性，包括自定义的脚本块或者引用第三方的脚本块。如果在 Scripts 中的<asp:ScriptReference />标签中设置了 Path 属性，它将覆盖该属性
OnAsyncPostBackError 方法	异步回传发生异常时的服务端处理函数，在这里可以捕获异常信息并作相应的处理
OnResolveScriptReference 方法	指定 ResolveScriptReference 事件的服务器端处理函数，在该函数中可以修改某一条脚本的相关信息如路径、版本等

10.2.2　ScriptManager 控件的用法

下面是一个简单的 ASP.NET AJAX 示例，实现将 DropDownList 已选择值，传递给一个 Label 控件的 Text 属性。

【例 10-1】简单的 ASP.NET AJAX 示例。

(1) 创建名称为 WebSite10 的网站。

(2) 在 WebSite10 网站中添加一个名为 Example.aspx 的【Web 窗体】网页。

(3) 将 1 个 ScriptManager 控件和 1 个 UpdatePanel 控件拖动到 Example.aspx 的【设计】视图中，DropDownList 和 Label 控件。

(4) 给 DropDownList 控件增加 2 个选择项：Hello World 和 Hello Ajax，同时设置 DropDownList 控件的 AutoPastBack 属性为 True。此时网页【设计】视图如图 10-1 所示。

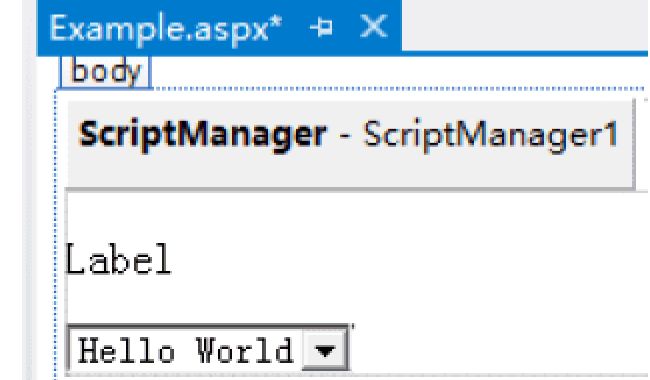


图 10-1　Example.aspx 的设计视图

对应【源】视图中的部分代码如下。

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
            <ContentTemplate>
                <br />
                <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
                <br />
                <br />
                <asp:DropDownList ID="DropDownList1" runat="server">
                    <asp:ListItem>Hello World</asp:ListItem>
                    <asp:ListItem>Hello Ajax</asp:ListItem>
                </asp:DropDownList>
            </ContentTemplate>
        </asp:UpdatePanel>
```

(5) 双击【设计】视图中的 DropDownList 控件，添加如下后台代码。

```
protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
```

```
    {
        Label1.Text = DropDownList1.SelectedValue;
    }
```

(6) 按 Ctrl+F5 组合键运行，选择 DropDownList 控件的选项，观察页面刷新效果。此时可以看到，浏览器的【后退】按钮是灰色的，该示例不像其他程序那样出现屏幕闪烁，说明没有进行整个页面的更新。

10.3　UpdatePanel 控件

UpdatePanel 控件是 ASP.NET AJAX 中很重要的一个控件，它可以用来创建局部更新的 Web 应用程序。有了 UpdatePanel 控件，开发者不需要编写任何客户端脚本，只须在页面上添加 UpdatePanel 控件和 ScriptManager 控件就可以自动实现局部更新。

10.3.1　UpdatePanel 控件工作原理

UpdatePanel 控件的工作依赖于 ScriptManager 控件和客户端 PageRequestManager 类。当 ScriptManager 允许页面局部更新时，它会以异步的方式回传给服务器。与传统的整页回传方式不同的是，只有包含在 UpdatePanel 中的页面部分会被更新，在从服务器返回 XHTML 之后，PageRequestManager 会通过操作 DOM 对象来替换需要更新的代码片段。UpdatePanel 工作过程如图 10-2 所示。

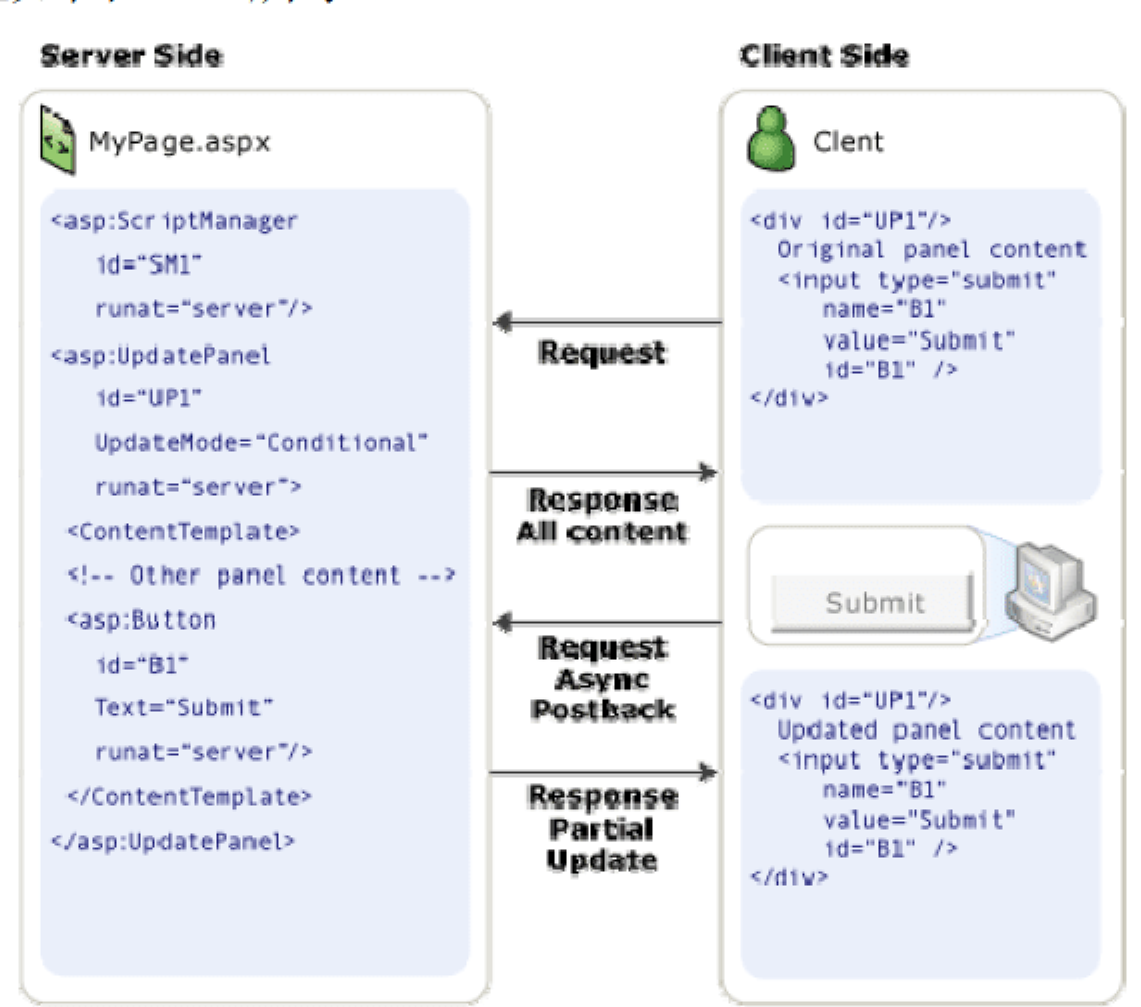


图 10-2　UpdatePanel 工作过程

从图 10-2 中可以看出，当客户端向服务器第一次发出请求时，服务器返回整个页面。另外，均通过异步回传方式对页面进行局部更新。

10.3.2　UpdatePanel 控件属性

UpdatePanel 控件的常用属性如表 10-2 所示。

表 10-2　UpdatePanel 控件的常用属性

属　性	说　明
ContenteTemplate	定义 UpdatePanel 的内容
Triggers	定义控件的服务器事件为异步或同步触发器
ChildrenAsTriggers	当 UpdateMode 属性为 Conditional 时，UpdatePanel 中的子控件的异步回送是否会引发 UpdatePanle 的更新
RenderMode	表示 UpdatePanel 最终呈现的 HTML 元素。Block(默认)表示<div>，Inline 表示<span>
UpdateMode	表示 UpdatePanel 的更新模式，有 2 个选项：Always 和 Conditional。Always 是不管有没有 Trigger，其他控件都将更新该 UpdatePanel，Conditional 表示只有当前 UpdatePanel 的 Trigger，或 ChildrenAsTriggers 属性为 true 时当前 UpdatePanel 中控件引发的异步回送或者整页回送，或是服务器端调用 Update()方法才会引发更新该 UpdatePanel

ContenteTemplate 和 Triggers 都是 UpdatePanel 控件的重要属性。

1. ContentTemplate 属性

ContenteTemplate 属性用来定义 UpdatePanel 的内容，其内容包括其他控件或 HTML 元素。例如，在 UpdatePanel 的内容放置 1 个 Label 控件和 1 个 Image 控件，代码如下。

```
……
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
              <asp:Label ID="Label1" runat="server" Text="Label"> </asp:Label>
              <asp:Image ID="Image1" runat="server" />
        </ContentTemplate>
</asp:UpdatePanel>
……
```

2. Triggers 属性

在 UpdatePanel 中有 2 种 Triggers，分别为 AsyncPostBackTrigger 和 PostBackTrigger。AsyncPostBackTrigger 用来指定某个服务器控件作为该 UpdatePanel 的异步更新触发器，PostBackTrigger 用来指定在 UpdatePanel 中的某个服务器控件作为同步更新触发器。异步更新触发器所引发的回传叫异步回传，引发页面局部更新。同步更新触发器所引发的回传叫同步回传，引发传统的整页回传。

例如，Button1 采用异步回传方式，而 Button2 采用同步回传方式，代码如下。

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
          ……
        </ContentTemplate>
<Triggers>
              <asp:AsyncPostBackTrigger ControlID="Button1"   />
              <asp:PostBackTrigger ControlID="Button2" />
```

```
        </Triggers>
    </asp:UpdatePanel>
```

10.3.3　在 UpdatePanel 中实现局部更新

在一个页面中，如果需要局部更新的内容较少，可以放置 1 个 UpdatePanel 控件，在该控件内实现局部更新的效果。

1．在 UpdatePanel 中实现局部更新

下面通过【例 10-2】介绍通过 UpdatePanel 控件实现局部更新的方法。

【例 10-2】通过 UpdatePanel 控件实现局部更新。

(1) 启动 VS，打开名称为 WebSite10 的网站。

(2) 新建名为 UpdatePanel_test.aspx 的网页，在其【设计】视图中，放置 ScriptManager、UpdatePanel 两个控件。

(3) 在 UpdatePanel 内部放入 2 个 Label 控件和 1 个 Button 控件，同时在 UpdatePanel 外面也放置 1 个 Button 控件。

该页面的【设计】视图如图 10-3 所示。

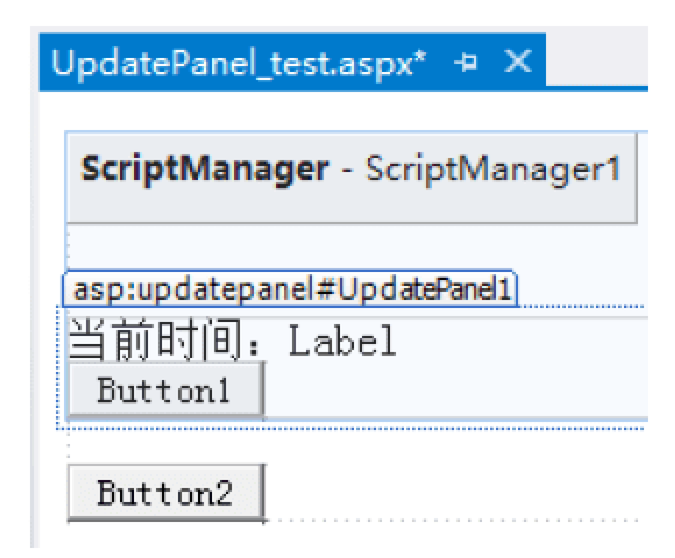


图 10-3　UpdatePanel_test.aspx 的设计页面

对应【源】视图中的代码如下。

```
    <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
            <br />
            <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                <ContentTemplate>
                    <asp:Label ID="Label1" runat="server" Text="当前时间："></asp:Label>
                    <asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>
                    <br />
                    <asp:Button ID="Button1" runat="server" Text="Button1" />
                </ContentTemplate>
            </asp:UpdatePanel>
        </div>
            <br />
            <asp:Button ID="Button2" runat="server" Text="Button2" />
```

(4) 分别双击【设计】视图中 Button 控件，添加如下所示的后台代码。

```
    protected void Button1_Click(object sender, EventArgs e)
    {
        this.Label2.Text = DateTime.Now.ToString();
    }
    protected void Button2_Click(object sender, EventArgs e)
    {
        this.Label2.Text = DateTime.Now.ToString();
    }
```

(5) 按 Ctrl+F5 组合键运行，单击 Button1 和 Button2 按钮，观察有什么不同。

可以看到，虽然单击 Button1 和 Button2 按钮都实现了显示时间的 Label 控件更新，但页面刷新效果却不同。通常默认情况下，在 UpdatePanel 内部的服务器控件采用的是异步回传方式，实现 UpdatePanel 的局部更新，而在 UpdatePanel 外面的服务器控件采用的是同步回传方式，实现整个页面的刷新。

2. 利用 Triggers 属性指定服务器控件回传方式

下面通过对【例 10-2】简单的修改，介绍一下如何利用 Triggers 属性指定服务器控件回传方式。

【例 10-3】利用 Triggers 属性指定服务器控件的回传方式。

(1) 启动 VS，打开 webSite10 网站。

(2) 在 UpdatePanel_test.aspx 的【源】视图中增加 Triggers 标记，指定 Button1 控件为同步触发器，Button2 控件为异步触发器。代码如下。

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <ContentTemplate>
        <asp:Label ID="Label1" runat="server" Text="当前时间："></asp:Label>
        <asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>
        <br /> <br />
        <asp:Button ID="Button1" runat="server" Text="Button1" onclick="Button1_Click" />
        <br />
    </ContentTemplate>
    <Triggers>
          <asp:PostBackTrigger ControlID="Button1" />
          <asp:AsyncPostBackTrigger ControlID="Button2"   />
  </Triggers>
 </asp:UpdatePanel>
<br />
<asp:Button ID="Button2" runat="server" Text="Button2" onclick="Button2_Click" />
```

(3) 按 Ctrl+F5 组合键运行。

此时可以看到，虽然 Button1 按钮在 UpdatePanel 内部，但实现的是整个页面的更新，而在 UpdatePanel 外面的 Button2 按钮却实现了 UpdatePanel 局部更新，究其原因是 Triggers 属性指定回传方式的结果。

10.3.4　同一页面上使用多个 UpdatePanel

使用 UpdatePanel 的时候并没有限制在一个页面上用多少个 UpdatePanel，所以可以为不同的需要局部更新的页面区域加上不同的 UpdatePanel。由于 UpdatePanel 默认的 UpdateMode 是 Always，如果页面上有一个局部更新被触发，则所有的 UpdatePanel 都将更新，有时这是程序员不愿看到的。这时只需要 UpdatePanel 在它自己的触发器触发的时候更新就可以了，所以需要把 UpdateMode 设置为 Conditional。

下面的【例 10-4】包括 2 个 UpdatePanel，其中一个用来输入数据，而另一个则用来显示数据，每一个 UpdatePanel 的 UpdateMode 属性都设置为 Conditional，当单击【新增】按钮时，2 个 UpdatePanel 都更新，当单击【取消】按钮时，只有用来数据输入的那个 UpdatePanel 更新。

【例 10-4】在同一页面上使用多个 UpdatePanel 控件。

(1) 启动 VS，打开 webSite10 网站，新建 multiUpdatePanel.aspx 页面。

(2) 在 multiUpdatePanel.aspx 的【设计】视图中，放置 1 个 ScriptManager 和 2 个 UpdatePanel 控件。

(3) 在一个 UpdatePanel 中放入 1 个 ListBox 控件和 1 个 Label 控件，而在另一个 UpdatePanel 中放置 1 个 TextBox 控件、1 个 Label 控件和 2 个 Button 控件。

该页面的【设计】视图如图 10-4 所示。

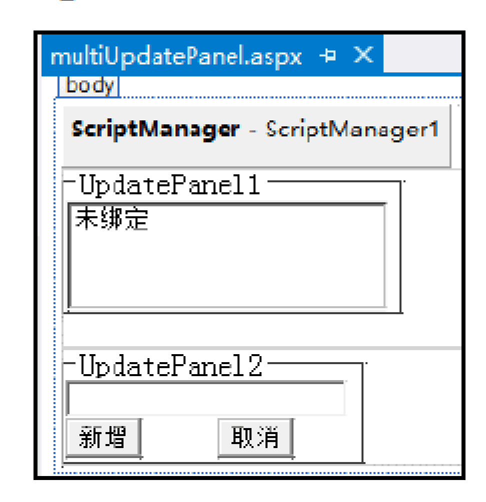


图 10-4　multiUpdatePanel.aspx 的设计页面

对应【源】视图中的部分代码如下。

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
      ……
<asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Conditional">
   <ContentTemplate>
      <fieldset style="width:180px;">
      <legend >UpdatePanel1</legend>
      <asp:ListBox ID="ListBox1" runat="server"    Width="170px"></asp:ListBox>
      <asp:Label ID="Label1" runat="server" ><%=DateTime.Now %></asp:Label>
      </fieldset>
   </ContentTemplate>
   <Triggers>
        <asp:AsyncPostBackTrigger ControlID="InsertButton" EventName="Click" />
   </Triggers>
  </asp:UpdatePanel>
• • •
<asp:UpdatePanel ID="UpdatePanel2" runat="server" UpdateMode="Conditional">
   <ContentTemplate>
      <fieldset style="width:160px;">
   <legend >UpdatePanel2</legend>
   <asp:TextBox ID="TextBox1" runat="server" EnableTheming="True"></asp:TextBox>
<asp:Button ID="InsertButton" runat="server" Text="新增"    OnClick="InsertButton_Click" />
  
   <asp:Button ID="Cancelbutton" runat="server" Text="取消" OnClick="Cancelbutton_Click" />
        <asp:Label ID="Label2" runat="server" ><%=DateTime.Now %></asp:Label>
   </fieldset>
  </ContentTemplate>
</asp:UpdatePanel>
```

(4) 分别双击【设计】视图中 Button 控件，添加如下所示的后台代码。

```
protected void InsertButton_Click(object sender, EventArgs e)
{
```

```
            ListBox1.Items.Add(TextBox1.Text);
            TextBox1.Text = String.Empty;
      }
      protected void Cancelbutton_Click(object sender, EventArgs e)
      {
           TextBox1.Text = String.Empty;
      }
```

(5) 按 Ctrl+F5 组合键运行，程序运行效果如图 10-5 所示。

此时可以看到，单击【新增】按钮会同时更新 UpdatePanel1 和 UpdatePanel2，而单击【取消】按钮则只会更新 UpdatePanel2。

【例 10-4】说明了如何在一个页面中使用多个 UpdatePanel，并将其 UpdateMode 设置为 Conditional 以及利用 Triggers 避免异步更新时多个 UpdatePanel 同时更新的情况。

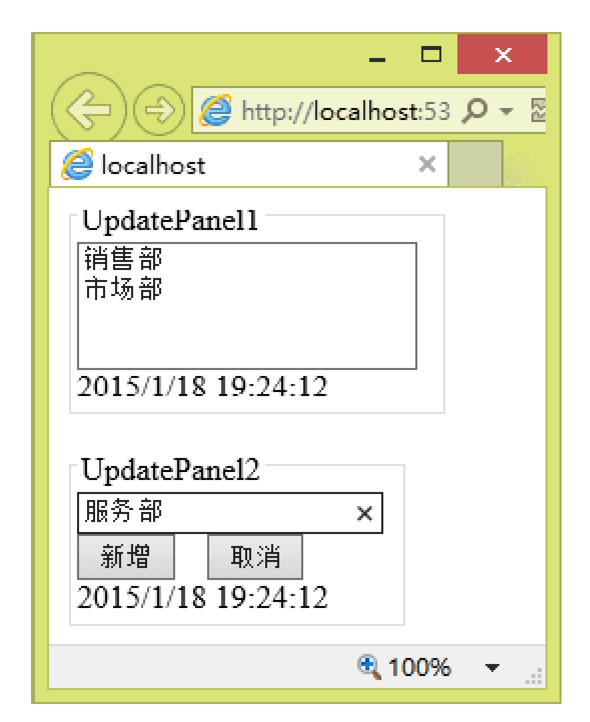


图 10-5　multiUpdatePanel.aspx 的运行效果

10.3.5　在内容页中使用 UpdatePanel

由于网页发送到客户端时，母版页和内容页的内容是合并在一起的，并且网页只能有一个 ScriptManager 控件，因此只需要在母版页中放置 ScriptManager 控件，内容页就可以使用 UpdatePanel 了。

1. 在内容页中使用 UpdatePanel

下面通过具体示例介绍在内容页中使用 UpdatePanel 控件的方法。

【例 10-5】在内容页中使用 UpdatePanel 控件的示例。

(1) 启动 VS，打开 webSite10 网站。

(2) 在【解决方案资源管理器】中，添加一个名为 MasterPage.master 的母版页。

(3) 在工具箱的【AJAX 扩展】标签下双击 ScriptManager 控件，将 ScriptManager 控件添加到母版页中，确保 ScriptManager 控件添加在 ContentPlaceHolder 控件之外，如图 10-6 所示。

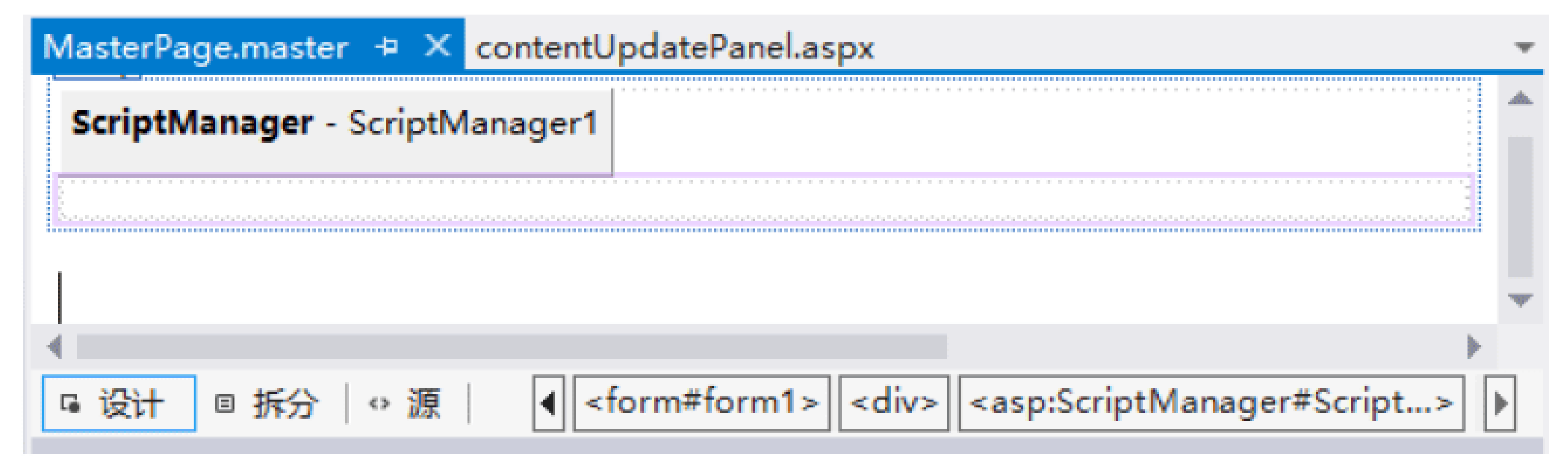


图 10-6　MasterPage.master 母版页设计视图

(4) 以 MasterPage.master 为母版，创建一个名为 ContentPage.aspx 的内容页。

(5) 在 ContentPage.aspx 页面中添加 1 个 UpdatePanel 控件，并在 UpdatePanel 控件中

放置 1 个 Calendar 控件，如图 10-7 所示。

(6) 按 Ctrl＋F5 组合键运行，当单击 Calendar 控件中的链接时，页面进行局部刷新。

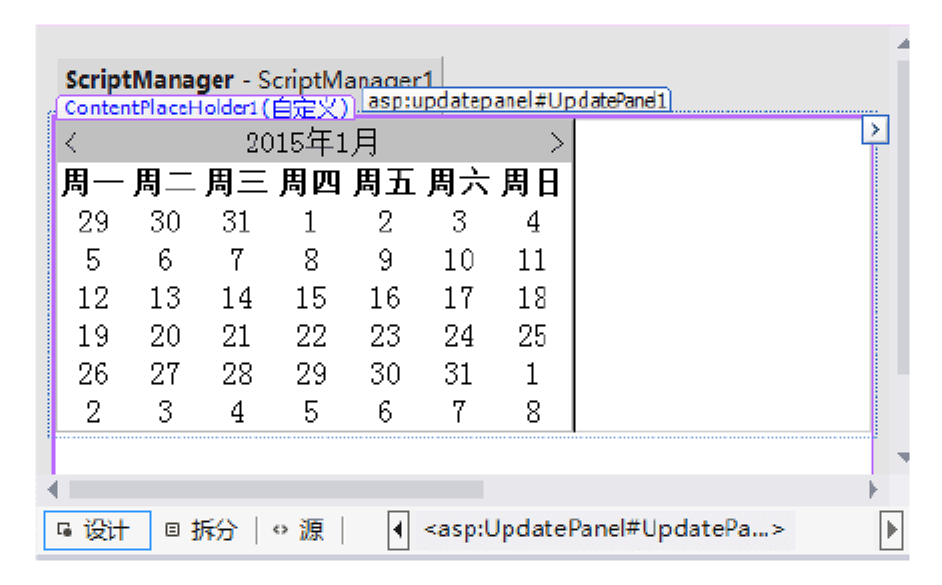


图 10-7　ContentPage. aspx 内容页设计视图

2. 从母版页更新内容页的 UpdatePanel

下面具体通过示例介绍如何从母版页更新内容页的 UpdatePanel。

【例 10-6】在母版页中更新内容页的 UpdatePanel 的示例。

(1) 启动 VS，打开 Website10 网站。

(2) 在【解决方案资源管理器】中，添加一个名为 MasterPage1.master 的母版页。

(3) 将母版页 HTML 代码改写为如下内容。

```
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server" />
        <asp:Button ID="IncrementButton" runat="server" Text="后一天"
            onclick="IncrementButton_Click" />
        <asp:Button ID="DecrementButton" runat="server" Text="前一天"
            onclick="DecrementButton_Click" />
        <asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">
        </asp:ContentPlaceHolder>
    </div>
    </form>
</body>
```

(4) 以 MasterPage1.master 为母版，创建一个名为 ContentPage1.aspx 的内容页。

(5) 将内容页 HTML 代码改写为如下内容。

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <ContentTemplate>
        <asp:Calendar ID="Calendar1" runat="server"></asp:Calendar>
    </ContentTemplate>
    <Triggers>
        <asp:AsyncPostBackTrigger ControlID="IncrementButton" />
        <asp:AsyncPostBackTrigger ControlID="DecrementButton" />
    </Triggers>
</asp:UpdatePanel>
</asp:Content>
```

(6) 将 MasterPage1.master 后台代码改写为如下内容。

```
protected void IncrementButton_Click(object sender, EventArgs e)
{
```

```
            //找到内容页中的 Calendar1 对象
                Calendar cal = ((Calendar)ContentPlaceHolder1.FindControl("Calendar1"));
            //将 Calendar1 对象日期增加一天
                cal.SelectedDate = cal.SelectedDate.AddDays(1) ;
    }
    protected void DecrementButton_Click(object sender, EventArgs e)
    {
            //找到内容页中的 Calendar1 对象
                Calendar cal = ((Calendar)ContentPlaceHolder1.FindControl("Calendar1"));
            //将 Calendar1 对象日期减少一天
            cal.SelectedDate = cal.SelectedDate.AddDays(-1) ;
    }
```

(7) 将 ContentPage1.aspx 后台代码改写为如下内容。

```
    protected void Page_Load(object sender, EventArgs e)
    {
        //首次加载初始化 Calendar1 对象
        if (!IsPostBack)
            Calendar1.SelectedDate = DateTime.Today;
    }
```

(8) 按 Ctrl+F5 键运行程序，此时单击母版页中的【前一天】和【后一天】按钮，触发了内容页中 UpdatePanel 控件的局部刷新，程序运行效果如图 10-8 所示。



图 10-8　ContentPage1.aspx 的运行效果

10.3.6　UpdatePanel 异步更新中的错误处理

当发生 UpdatePanel 控件异步更新错误时，默认情况下会弹出一个错误对话框。如果设计者觉得不符合用户习惯，可以通过 ScriptManager 控件的 OnAsyncPostBackError　事件和 AsyncPostBackErrorMessage 属性捕捉和设置回传时的错误消息。

【例 10-7】将异步更新中的错误消息发送给客户端。

(1) 启动 VS，打开 WebSite10 网站，新建 UpdatePanel3.aspx。

(2) 在 UpdatePanel3.aspx 的【设计】视图中，放置一个 ScriptManager 和一个 UpdatePanel 控件。

(3) 该页面设计如图 10-9 所示。

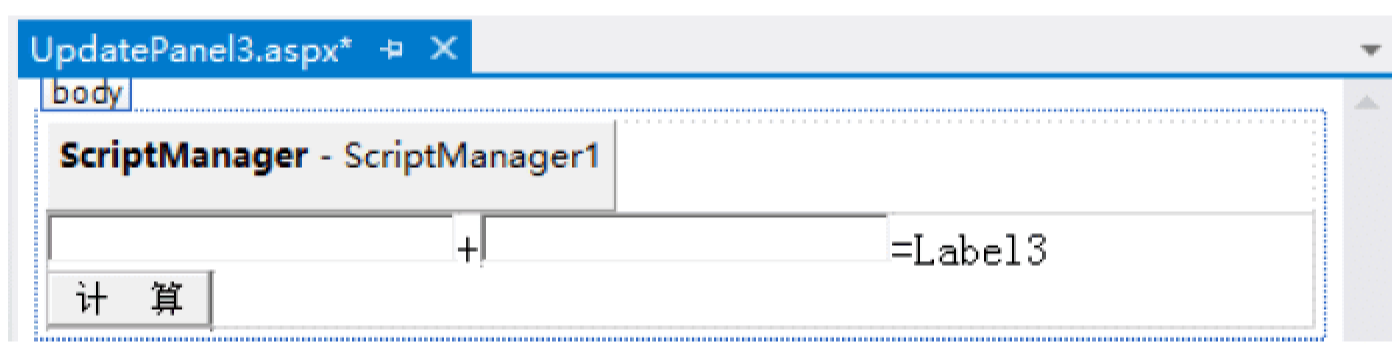


图 10-9　UpdatePanel3.aspx 的设计页面

对应【源】视图中的代码如下。

```
    <asp:ScriptManager ID="ScriptManager1" runat="server"
```

```
                onasyncpostbackerror="ScriptManager1_AsyncPostBackError">
</asp:ScriptManager>
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
<ContentTemplate>
  <asp:TextBox ID="TextBox1" runat="server" Width="80px"></asp:TextBox>
 <asp:Label ID="Label1" runat="server" Text="＋" ></asp:Label>
 <asp:TextBox ID="TextBox2" runat="server" Width="80px"></asp:TextBox>
 <asp:Label ID="Label2" runat="server" Text="＝"></asp:Label>
 <asp:Label ID="Label3" runat="server" Text="Label3"></asp:Label> <br /> <br />
 <asp:Button ID="Button1" runat="server" Text="计　算" onclick="Button1_Click" />
  </ContentTemplate>
</asp:UpdatePanel>
```

(4) 添加后台代码如下。

```
protected void Button1_Click(object sender, EventArgs e)
{
        try
        {
            Double a = Convert.ToDouble(TextBox1.Text);
            Double b = Convert.ToDouble(TextBox2.Text);
            Double res = a * b;
            Label3.Text = res.ToString("f2");
        }
        catch (Exception ex)
        {
            if (TextBox1.Text.Length >= 0 && TextBox2.Text.Length >= 0)
            {
                ex.Data["ExtraInfo"] = "这两个数无法相乘";
            }
            throw ex;
        }
}
protected void ScriptManager1_AsyncPostBackError(object sender, AsyncPostBackErrorEventArgs e)
{
        if (e.Exception.Data["ExtraInfo"] != null)
        {
             ScriptManager1.AsyncPostBackErrorMessage = e.Exception.Data["ExtraInfo"].ToString();
        }
        else
        {
             ScriptManager1.AsyncPostBackErrorMessage = e.Exception.Message;
        }

}
```

(5) 按 Ctrl＋F5 组合键运行，当相乘的两个数中有非法数字时，弹出自定义错误信息的对话框。

如果按 F5 键调试运行，调试器会首先捕捉到异常，从而中断代码的执行，所以要求程序在不调试的情况下运行。

10.4　UpdateProgress 控件

UpdateProgress 控件一般与 UpdatePanel 控件联合使用，即在 UpdatePanel 异步更新过程中，显示提示信息。这些信息可以是一段文字、进度条或各种动画。当异步更新完成时，提示信息自动消失。

10.4.1　UpdateProgress 控件属性

UpdateProgress 控件常用属性如表 10-3 所示。

表 10-3　UpdateProgress 控件常用属性

属　性	说　明
AssociateUpdatePanelID	设置哪个 UpdatePanel 控件产生的回送会显示 UpdateProgress 的内容
DisplayAfter	当引发回送后多少毫秒会显示 UpdateProgress 控件的内容
DynamicLayout	设置 UpdateProgress 控件的显示方式。true 当 UpdateProgress 控件不显示的时候不占用空间(默认)。false 当 UpdateProgress 控件不显示的时候仍然占用空间

如果没有设定UpdateProgress控件的AssociateUpdatePanelID属性，则任何一个异步更新都会使UpdateProgress控件显示出来。相反，如果将UpdateProgress控件的AssociateUpdatePanelID属性设为某个UpdatePanel控件的ID，那只有该UpdatePanel控件引发的异步更新才会使相关联的UpdateProgress控件显示出来。

10.4.2　使用 UpdateProgress 控件的方法

下面【例 10-8】页面上有 1 个 UpdatePanel 控件和 1 个 UpdateProgress 控件，当 UpdatePanel 控件异步更新时，显示 UpdateProgress 控件的提示内容。

【例 10-8】使用 UpdateProgress 控件的示例。

(1) 启动VS，打开WebSite10 网站，创建名称为UpdateProgressExample1.aspx的网页。

(2) 在 UpdateProgressExample1.aspx 的【设计】视图中，放置一个 ScriptManager、一个 UpdatePanel 和一个 UpdateProgress 控件。

(3) 在UpdatePanel中，添加一个Label控件和一个 Button 控件。

(4) 在 UpdateProgress 中，添加【正在更新】文字和进度条图像。

(5) 该页面的【设计】视图如图 10-10 所示。

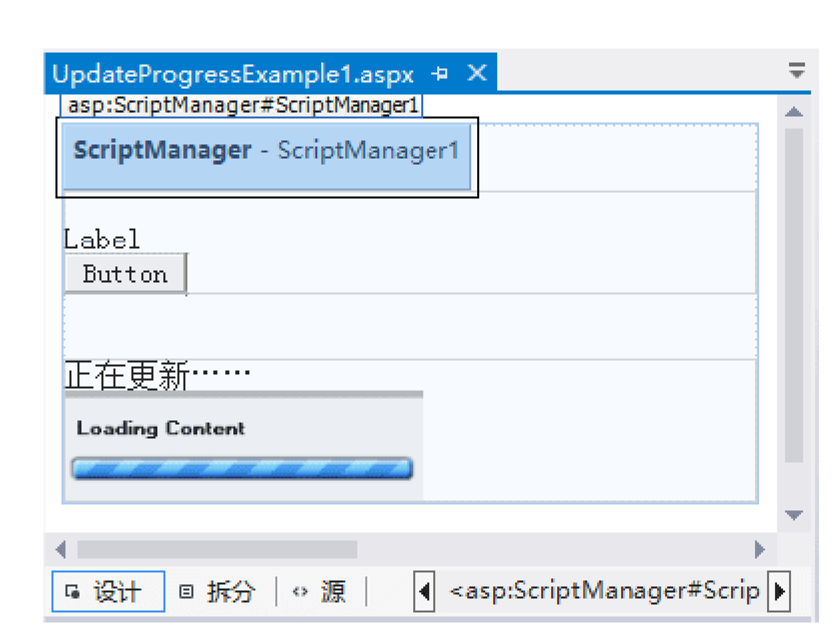


图 10-10　UpdateProgressExample1.aspx 的设计页面

对应【源】视图中的部分代码如下。

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
            <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>    <br />
            <asp:Button ID="Button1" runat="server" Text="Button" onclick="Button1_Click" />
            <br />
        </ContentTemplate>
    </asp:UpdatePanel>    <br /> <br />
    <asp:UpdateProgress ID="UpdateProgress1" runat="server">
        <ProgressTemplate>
          正在更新……<br />
        <asp:Image ID="Image1" runat="server" ImageUrl="progress.gif" Width="180px" /><br />
        </ProgressTemplate>
    </asp:UpdateProgress>
```

(6) 添加后台代码如下。

```
 protected void Button1_Click(object sender, EventArgs e)
{
    //延迟 3 秒
System.Threading.Thread.Sleep(3000);
    Label1.Text = DateTime.Now.ToString();
}
```

(7) 按 Ctrl+F5 键运行，观看程序运行效果，如图 10-11 所示。

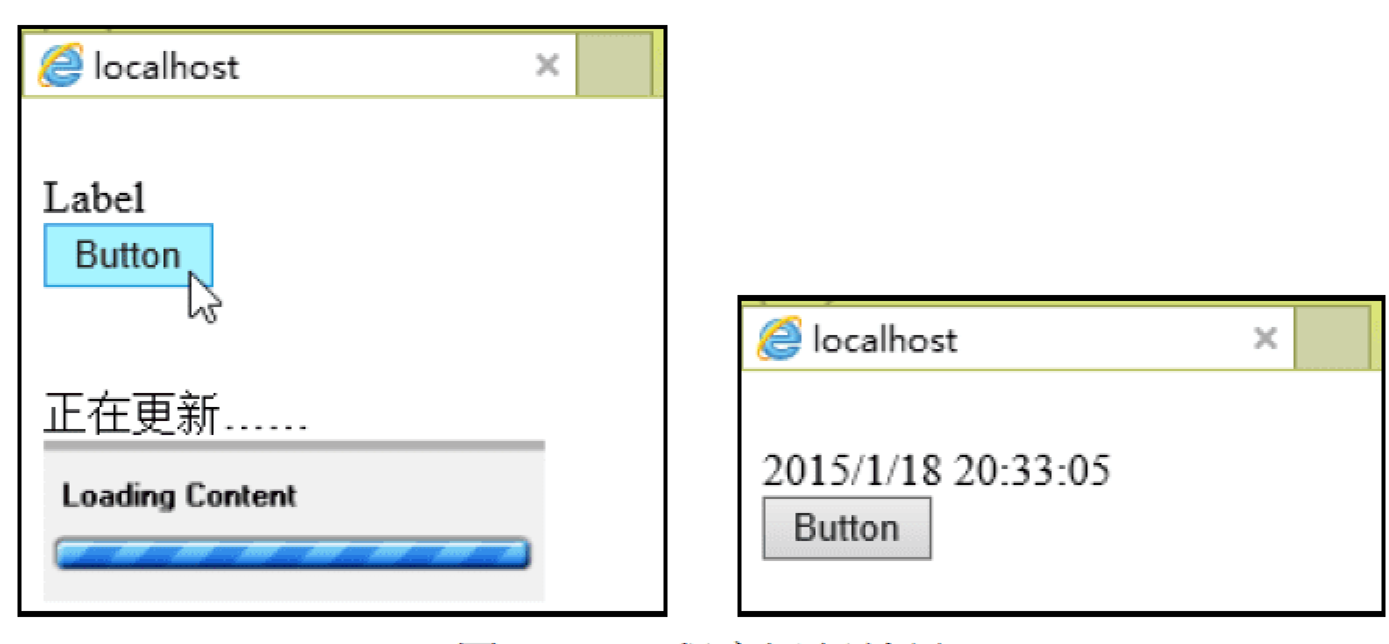


图 10-11　程序运行效果

10.4.3　使用多个 UpdateProgress 控件的方法

下面的【例 10-9】，演示了页面上使用多个 UpdateProgress 控件的方法，该示例的页面上放置两个 UpdatePanel 和两个 UpdateProgress 控件，将 UpdateProgress1 控件的 AssociateUpdatePanelID 属性设为 UpdatePanel1 控件的 ID，UpdateProgress2 控件的 AssociateUpdatePanelID 属性设为 UpdatePanel2 控件的 ID。当 UpdatePanel1 异步更新时，UpdateProgress1 显示，而当 UpdatePanel2 异步更新时，显示 UpdateProgress2 的提示信息。

【例 10-9】使用多个 UpdateProgress 控件。

(1) 启动 VS，打开 WebSite10 网站，创建名称为 UpdateProgressExample2.aspx 的网页。

(2) 在 UpdateProgressExample2.aspx 的【设计】视图中，放置一个 ScriptManager、两

个 UpdatePanel 和两个 UpdateProgress 控件。

(3) 在 UpdatePanel 控件和 UpdateProgress 控件中，添加与【例 10-8】一样的控件。

(4) 该页面设计如图 10-12 所示。

对应【源】视图中的部分代码如下。

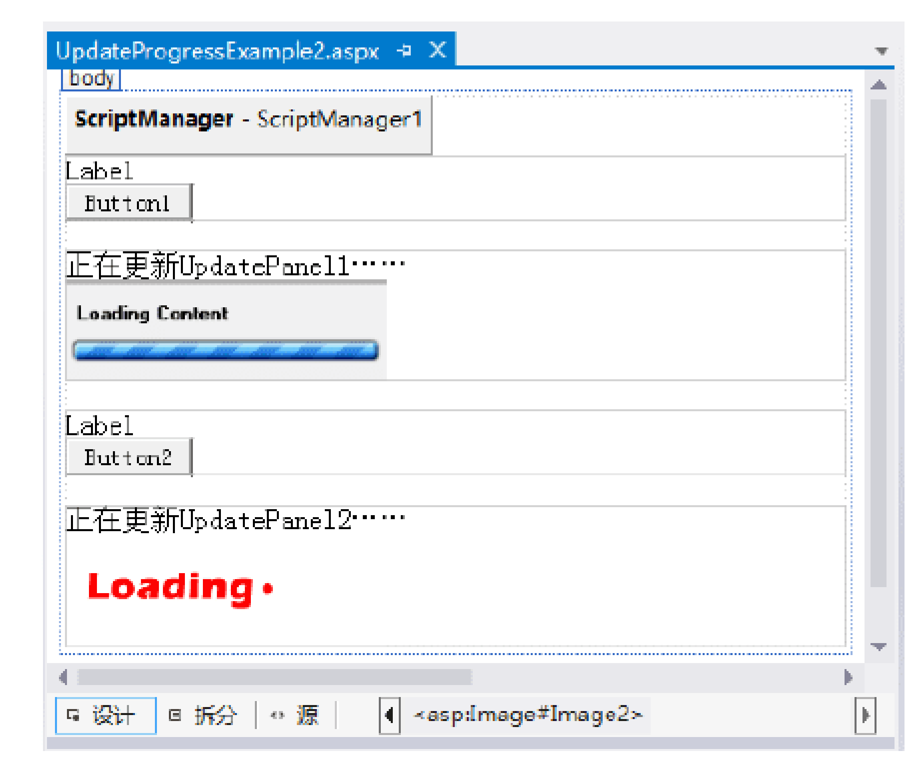


图 10-12　UpdateProgressExample2.aspx 的设计页面

```
    <asp:ScriptManager ID="ScriptManager1"
runat="server"></asp:ScriptManager>
    <asp:UpdatePanel ID="UpdatePanel1"
runat="server" UpdateMode="Conditional">
        <ContentTemplate>
        <asp:Label ID="Label1" runat="server"
Text="Label"></asp:Label>    <br />
        <asp:Button ID="Button1" runat="server" Text="Button1" onclick="Button1_Click" /><br />
        </ContentTemplate>
    </asp:UpdatePanel>    <br />
    <asp:UpdateProgress ID="UpdateProgress1" runat="server"
AssociatedUpdatePanelID="UpdatePanel1">
        <ProgressTemplate>
            正在更新 UpdatePanel1……<br />
            <asp:Image ID="Image1" runat="server" ImageUrl="strip.gif" Width="180px" /><br />
        </ProgressTemplate>
    </asp:UpdateProgress>     <br />
    <asp:UpdatePanel ID="UpdatePanel2" runat="server" UpdateMode="Conditional">
        <ContentTemplate>
        <asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>     <br />
        <asp:Button ID="Button2" runat="server" Text="Button2" onclick="Button2_Click" /> <br
/>
        </ContentTemplate>
    </asp:UpdatePanel>    <br />
    <asp:UpdateProgress ID="UpdateProgress2" runat="server"
AssociatedUpdatePanelID="UpdatePanel2">
        <ProgressTemplate>
            正在更新 UpdatePanel2……<br />
            <asp:Image ID="Image2" runat="server" ImageUrl="strip.gif" Width="180px" /><br />
        </ProgressTemplate>
    </asp:UpdateProgress>
```

(5) 添加后台代码如下。

```
protected void Button1_Click(object sender, EventArgs e)
{
    System.Threading.Thread.Sleep(3000);
    Label1.Text = DateTime.Now.ToString();
}
protected void Button2_Click(object sender, EventArgs e)
{
    System.Threading.Thread.Sleep(3000);
    Label2.Text = DateTime.Now.ToString();
}
```

(6) 按 Ctrl＋F5 键运行，可以观看异步更新时 UpdateProgress 控件的显示效果。

虽然一个页面允许有多个 UpdateProgress 控件，但是在实际中，一般在一个页面中应该只放置一个 UpdateProgress。

10.5　Timer 控件

Timer 控件是 ASP.NET AJAX 中又一个重要的服务器控件。通过它可以完成局部页面的定时更新，从而实现局部页面定时刷新、图片自动播放、超时自动退出等功能。

10.5.1　Timer 控件属性和事件

Timer 控件常用属性和事件如表 10-4 所示。

表 10-4　Timer 控件常用属性和事件

属性和事件	说　明
Interval 属性	用于指定间隔时间
Tick 事件	指定间隔到期后触发
Enabled 属性	用于表示是否允许 tick 事件

Interval 属性用来决定每隔多长时间要引发回传，其设置值的单位是毫秒，默认值则是 60000ms，也就是 60s。需要注意的是，将 Timer 控件的 Interval 属性设置成较小的值会使得回送频率增加，也很容易使得 Web 服务器的流量大增，对整体资源耗用与效率都会造成不良的影响。因此，尽量在必要的时候使用 Timer 控件来定时更新页面上的内容。

每当 Timer 控件的 Interval 属性所设置的间隔时间到达而进行回传时，就会在服务器上引发 Tick 事件，执行定时操作。

另外一个常用的属性就是 Enabled，用户可以将 Enabled 属性设置成 false 以便让 Timer 控件停止计时，而当需要让 Timer 控件再次开始计时的时候，只须再将 Enabled 属性设置成 True 即可。

Timer 控件在 UpdatePanel 控件的内外是有区别的。当 Timer 控件在 UpdatePanel 控件内部时，JavaScript 计时组件只有在一次回传完成后才会重新建立。也就是说直到网页回传完成之前，定时器间隔时间不会从头计算。例如，用户设置 Timer 控件的 Interval 属性值为 3000ms(3s)，但是回传操作本身却花了 2s 才完成，则下一次的回传将发生在前一次回传被引发之后的 5s。而如果 Timer 控件位于 UpdatePanel 控件之外，则当回传正在处理时，下一次回传将发在前一次回传被引发之后的 3s。也就是说，UpdatePanel 控件的内容被更新之后的 1s，就会再次看到该控件被更新。

10.5.2　使用 Timer 控件定时更新 UpdatePanel

下面介绍一个在 UpdatePanel 内部使用 Timer 控件的简单示例，该示例实现图片的自动刷新。

【例 10-10】在 UpdatePanel 内部使用 Timer 控件。

(1) 启动 VS，打开 WebSite10，创建名称为 TimerExample1.aspx 的网页。

(2) 在 TimerExample1.aspx 的【设计】视图中，放置一个 ScriptManager、一个 UpdatePanel。

(3) 在 UpdatePanel 控件中，添加一个 Timer 控件和一个 Image 控件。

(4) 该页面的设计如图 10-13 所示。

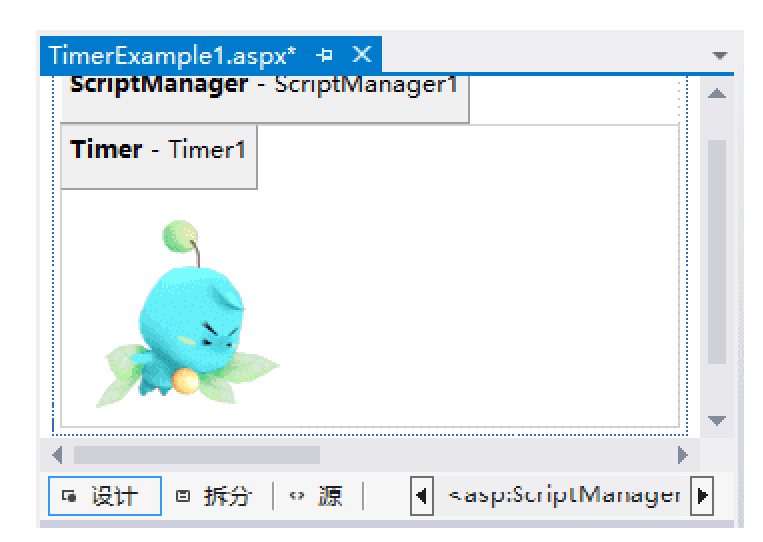


图 10-13　TimerExample1.aspx 的设计页面

对应【源】视图中的部分代码如下。

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
            <asp:Timer ID="Timer1" runat="server" Interval="1000" ontick="Timer1_Tick">
            </asp:Timer>
            <asp:Image ID="Image1" runat="server" ImageUrl="TimerExample1.gif"/>
        </ContentTemplate>
    </asp:UpdatePanel>
```

(5) 添加后台代码如下。

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Page.IsPostBack == false)
    {
        //设置网页上的变量
ViewState["number"] = 1;
    }
}
protected void Timer1_Tick(object sender, EventArgs e)
{
    ViewState["number"] = (int)ViewState["number"] % 4 + 1;
    Image1.ImageUrl = string.Format("~/images/{0}.png", ViewState["number"]);
}
```

(6) 按 Ctrl+F5 组合键运行，可以看到每间隔 1s 局部刷新一幅图片。

10.5.3　使用 Timer 控件定时更新多个 UpdatePanel

在下面的【例 10-11】中，Timer 控件定时更新 2 个 UpdatePanel 控件，Timer 控件被放在 UpdatePanel 控件的外面，并将它配置为 UpdatePanel 的触发器。

【例 10-11】使用 Timer 控件定时更新多个 UpdatePanel 示例。

(1) 启动 VS，打开 WebSite10，创建名称为 TimerExample2.aspx 页面。

(2) 在 TimerExample2.aspx 的【设计】视图中，放置一个 ScriptManager、两个 UpdatePanel。

(3) 在所有 UpdatePanel 控件之外，添加一个 Timer 控件，并设置它为 UpdatePanel 的异步触发器。

(4) 该页面设计如图 10-14 所示。

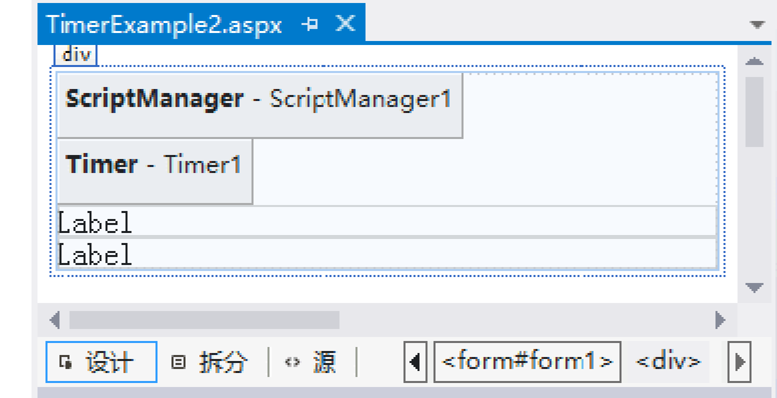


图 10-14　TimerExample2.aspx 的设计页面

对应【源】视图中的部分代码如下。

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
<asp:Timer ID="Timer1" runat="server" Interval="2000" ontick="Timer1_Tick">
</asp:Timer>
<asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Conditional">
        <ContentTemplate>
                <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
        </ContentTemplate>
        <Triggers>
              <asp:AsyncPostBackTrigger ControlID="Timer1" />
        </Triggers>
</asp:UpdatePanel>
<asp:UpdatePanel ID="UpdatePanel2" runat="server" UpdateMode="Conditional">
        <ContentTemplate>
              <asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>
        </ContentTemplate>
        <Triggers>
              <asp:AsyncPostBackTrigger ControlID="Timer1" />
        </Triggers>
</asp:UpdatePanel>
```

(5) 添加后台代码如下。

```
protected void Timer1_Tick(object sender, EventArgs e)
{
     Label1.Text = "UpdatePanel1  更新于：  " +  DateTime.Now.ToLongTimeString();
     Label2.Text = "UpdatePanel2  更新于：  " +  DateTime.Now.ToLongTimeString();
}
```

(6) 按 Ctrl＋F5 组合键运行，可以看到每间隔 2s 同时刷新两个 UpdatePanel。

10.6　ASP.NET AJAX Control Extenders 扩展控件

ASP.NET AJAX Control Extenders 是一些派生自 System.Web.UI.ExtenderControl 基类的控件，通常后缀为 Extender，扩展控件必须和被其控制的控件组合才能发挥作用。

利用扩展控件可以为页面中已存在的控件添加其他功能(一般都是 AJAX 或者

JavaScript 支持)。它们使得开发者可以封装控件行为，并且使得为应用程序添加更丰富的功能变得非常简单。

如图 10-15 所示，利用 CalendarExtender 日期扩展控件，使得 TextBox 控件录入日期更加方便、直观，而且增加了 AJAX 功能。



图 10-15　TextBox 和 CalendarExtender 组合

TextBox 和 CalendarExtender 组合的代码如下。

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<cc1:CalendarExtender ID="TextBox1_CalendarExtender" runat="server"
        Enabled="True" TargetControlID="TextBox1">
 </cc1:CalendarExtender>
```

其中将 CalendarExtender 控件的 TargetControlID 属性指向 TextBox 控件的 ID。

10.6.1　使用 ASP.NET AJAX Control Extenders 的方法

ASP.NET AJAX Control Toolkit 是一个非常好的扩展控件工具包，是由 Microsoft 公司和其他开发人员共同开发的，它包括四十多个免费的扩展控件，而且一直都在增加，可以轻松地被下载并添加到 VS 的工具箱。

1. 添加扩展控件到 VS 工具箱

在 VS 工具箱中添加扩展控件的步骤如下。

(1) 从 Microsoft 公司网站上，下载 ASP.NET AJAX Control Toolkit For .NET 4.5 工具包。

(2) 将该工具包解压缩。

(3) 在 VS 工具箱中，新建一个名为 AJAX Control Toolkit 的选项卡。

(4) 在解压后的工具包中找到 AJAXControlToolkit.dll 文件，将其拖动到 VS 工具箱的 AJAX Control Toolkit 选项卡中。

这样，扩展控件就会出现在 VS 的工具箱中，如图 10-16 所示。

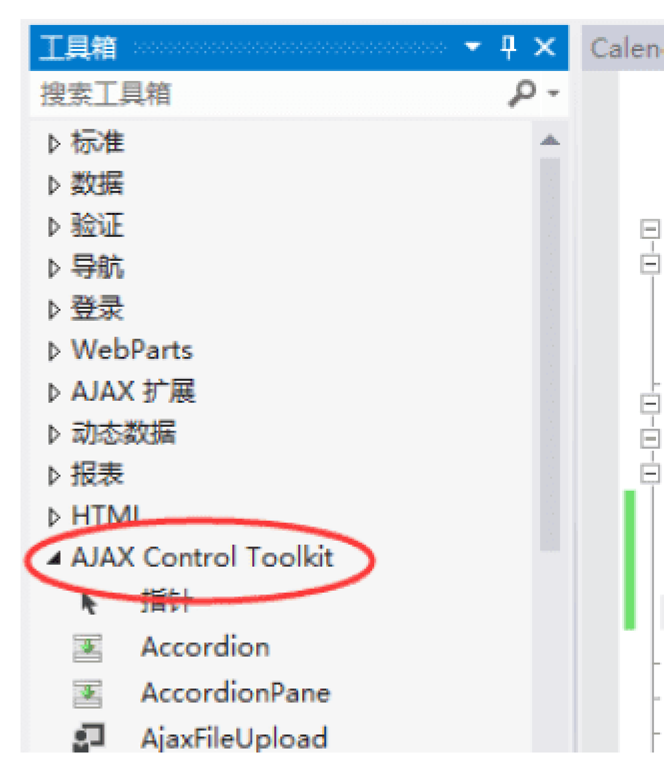


图 10-16　VS 的工具箱

2. 绑定扩展控件到某个已存在的控件

扩展控件被添加到 VS 工具箱后，再选择控件时，会发现一个新的【添加扩展程序】任务选项出现在被选择的控件上，如图 10-17 所示。

如果单击【添加扩展程序】任务选项，将会弹出如图 10-18 所示的对话框，它包含了所有可以选择的扩展控件，确定后就完成了扩展控件的绑定。

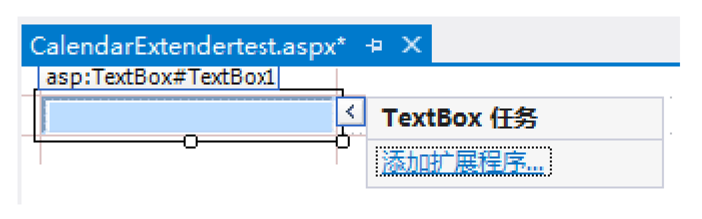


图 10-17　添加扩展控件

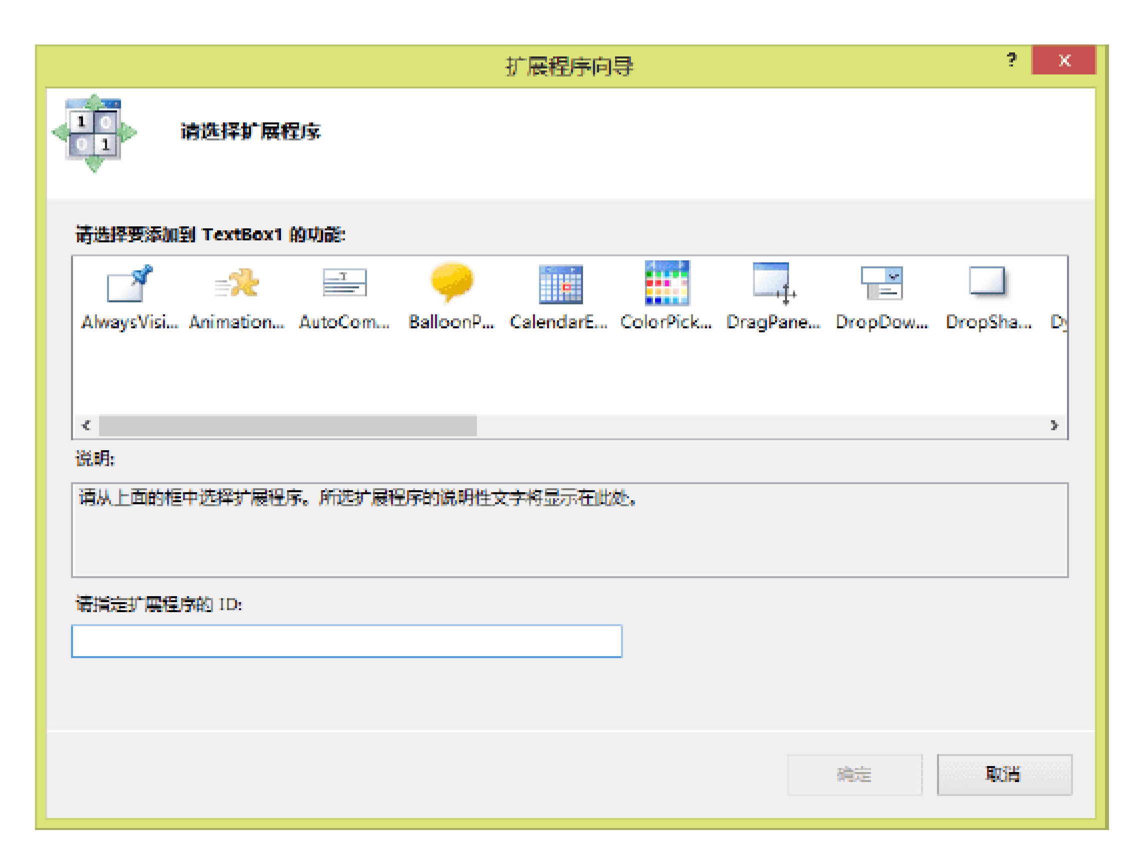


图 10-18　【选择扩展控件】对话框

3. 合并属性窗格

在 VS 中，当将一个扩展控件绑定到一个控件后，该控件的属性窗格也会扩展显示扩展控件的属性，如图 10-19 所示。

4. 删除绑定的扩展控件

如果想删除某个控件的扩展控件，只需要从控件的任务选项中选择【删除扩展程序】选项，如图 10-20 所示。

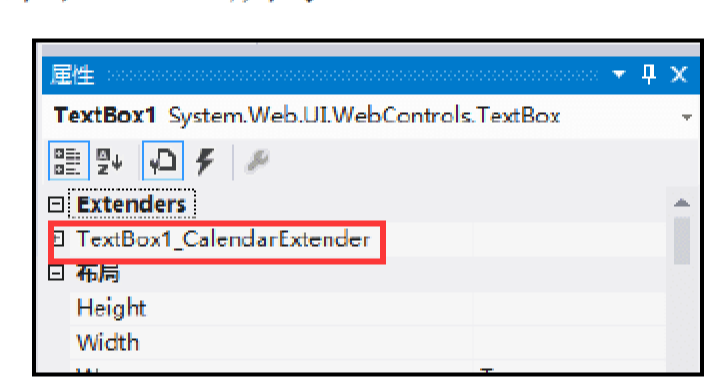


图 10-19　扩展显示属性窗格

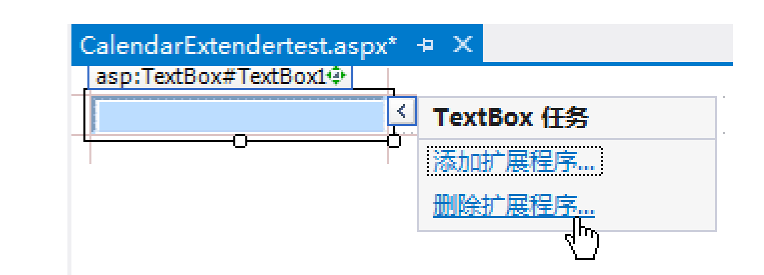


图 10-20　删除扩展控件

10.6.2　使用 FilteredTextBoxExtender 控件防止输入非法字符

FilteredTextBoxExtender 扩展控件是用来对文本框进行过滤的，让文本框只能对设定的值进行输入。输入类型有 Numbers(数字)，LowercaseLetters(小写字母)，UppercaseLetters(大写字母)和 Custom(自定义字符串)。FilteredTextBoxExtender 扩展控件的常用属性如表 10-5 所示。

表 10-5　FilteredTextBoxExtender 常用属性

属　　性	说　　明
TargetControlID	要进行过滤的目标 TextBox 的 ID
FilterType	字符过滤类型，提供的有如下 4 种：Numbers、LowercaseLetters、UppercaseLetters、Custom。它们之间可以同时指定多个类型，如 FilterType="Custom, Numbers"
ValidChars	当 FilterType 为 Custom 时允许输入的字符，否则将被忽略，如 ValidChars="+-=/*()."

下面通过具体示例演示 FilteredTextBoxExtender 扩展控件的用法。

【例 10-12】演示 FilteredTextBoxExtender 扩展控件的用法。

(1) 启动 VS，打开 WebSite10，创建名称为 FilteredTextBoxExtender.aspx 的网页。

(2) 在 FilteredTextBoxExtender.aspx 的【设计】视图中，放置一个 ScriptManager、一个 TextBox 和一个 Label。

(3) 单击 TextBox 控件的【添加扩展程序】任务选项，绑定 FilteredTextBoxExtender 扩展控件。

(4) 该页面设计如图 10-21 所示。

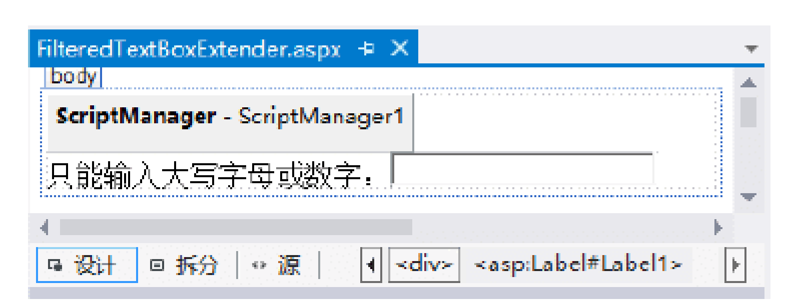


图 10-21　FilteredTextBoxExtender.aspx 的设计页面

对应【源】视图中的部分代码如下。

```
<%@ Register assembly="AjaxControlToolkit" namespace="AjaxControlToolkit" tagprefix="cc1" %>
• • •
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <asp:Label ID="Label1" runat="server" Text="只能输入大写字母或数字："></asp:Label>
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    <cc1:FilteredTextBoxExtender ID="TextBox1_FilteredTextBoxExtender"
  runat="server" Enabled="True" FilterType="Numbers,UppercaseLetters"
TargetControlID="TextBox1">
    </cc1:FilteredTextBoxExtender>
```

(5) 按 Ctrl+F5 组合键运行，可以看到 TextBox 控件只允许输入大写字母和数字。

由于使用文本框过滤是在客户端进行的，所以使用 FilteredTextBoxExtender 扩展控件的网页中必须包含 ScriptManager 控件。

10.6.3　使用 SlideShowExtender 控件播放幻灯片

SlideShowExtender 扩展控件用于控制 Image 控件，它可以很方便地实现自动和手动播放图片功能。由于播放图片是在客户端进行的，所以使用 SlideShowExtender 扩展控件的网页中必须包含 ScriptManager 控件。

SlideShowExtender 扩展控件的常用属性如表 10-6 所示。

表 10-6　SlideShowExtender 常用属性

属　　性	说　　明
TargetControlID	该控件的目标扩展控件
ImageDescriptionLabelID	对显示的图片进行说明的 Label 控件
Loop	是否为图片进行循环放映
NextButtonID	控制显示下一张图片的按钮

(续表)

属　性	说　明
PlayButtonID	控制进行播放或停止的按钮
PlayButtonText	当 Image 中的图片在放映时，PlayButtonID 按钮显示的文本
PreviousButtonID	控制显示前一张图片的按钮
StopButtonText	当 Image 中的图片停止放映时，PlayButtonID 按钮显示的文本
PlayInterval	播放每幅图片的间隔，单位 ms，默认 3000ms(3s)
SlideShowServiceMethod	进行幻灯片式放映时加载图片的方法

下面通过【例 10-13】演示 SlideShowExtender 扩展控件的用法。

【例 10-13】演示 SlideShowExtender 扩展控件的用法。

(1) 启动 VS，打开 WebSite10，创建名称为 SlideShowExtender.aspx 的网页。

(2) 在 SlideShowExtender.aspx 的【设计】视图中，放置一个 ScriptManager、1 个 Image、1 个 Label 和 3 个 Button 控件。Image 控件用于显示图片，Label 控件用于显示图片说明，Button 控件用于控制图片显示。

(3) 单击 Image 控件的【添加扩展程序】任务选项，绑定 SlideShowExtender 扩展控件。

(4) 该页面【源】视图中的部分代码如下。

```
<%@ Register assembly="AjaxControlToolkit" namespace="AjaxControlToolkit" tagprefix="cc1" %>
• • •
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <asp:Image ID="Image1" runat="server"
                    Height="300"
                    Style="border: 1px solid black;width:auto"
                    ImageUrl="~/images/Blue hills.jpg"
                    AlternateText="Blue Hills image" />
        <asp:Label runat="server" ID="imageDescription" ></asp:Label><br /><br />
        <asp:Button runat="Server" ID="prevButton" Text="前一个"   />
        <asp:Button runat="Server" ID="playButton" Text="开   始"   />
        <asp:Button runat="Server" ID="nextButton" Text="后一个"   />

        <cc1:SlideShowExtender   ID="slideshowextend1" runat="server"
                    TargetControlID="Image1"
                    SlideShowServiceMethod="GetSlides"
                    AutoPlay="true"
                    ImageDescriptionLabelID="imageDescription"
                    NextButtonID="nextButton"
                    PlayButtonText="开始"
                    StopButtonText="停止"
                    PreviousButtonID="prevButton"
                    PlayButtonID="playButton"
                    Loop="true" PlayInterval="2000">
       </cc1:SlideShowExtender>
```

(5) 添加后台代码如下。

```
[System.Web.Services.WebMethod]
[System.Web.Script.Services.ScriptMethod]
public static AjaxControlToolkit.Slide[] GetSlides()
{
  return new AjaxControlToolkit.Slide[] {
        new AjaxControlToolkit.Slide("images/Blue hills.jpg", "", "Blue hills"),
        new AjaxControlToolkit.Slide("images/Sunset.jpg", "", "Sunset"),
        new AjaxControlToolkit.Slide("images/Winter.jpg", "", "Winter"),
        new AjaxControlToolkit.Slide("images/Water lilies.jpg", "", "Lillies in the water"),
        new AjaxControlToolkit.Slide("images/VerticalPicture.jpg", "", "Portrait style icture")};
    }
}
```

GetSlides 方法创建了一个 AjaxControlToolkit.Slide 类型的数组，该数组包含了所有要播放的图片，SlideShowExtender 控件调用 GetSlides 方法，得到这些图片，并将其在 Images 控件中依次显示出来。

(6) 按 Ctrl＋F5 组合键运行，运行效果如图 10-22 所示。

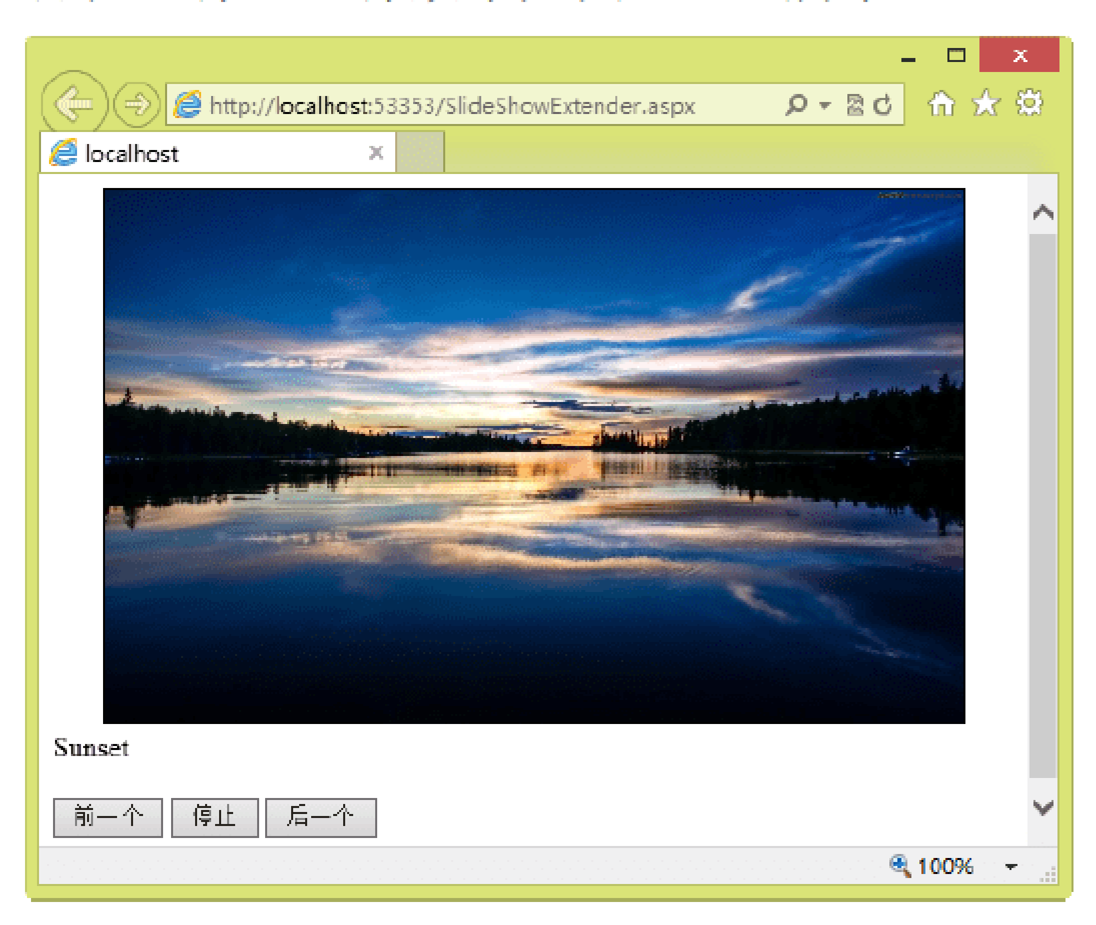


图 10-22　【例 10-13】的运行效果

10.6.4　使用 AlwaysVisibleControlExtender 固定位置显示控件

AlwaysVisibleControlExtender 扩展控件使其被控制的控件显示在页面的固定位置上，常用于漂浮在网页两边，跟随着滚动条滚动的悬浮广告等。AlwaysVisibleControlExtender 是一个非常简单的扩展控件，不用编码只需简单的设置就可使用。

AlwaysVisibleControlExtender 扩展控件的常用属性如表 10-7 所示。

表 10-7　AlwaysVisibleControlExtender 常用属性

属　　性	说　　明
TargetControlID	目标控件 ID，要浮动的控件
HorizontalOffset	距离浏览器的水平边距，默认值 0px
HorizontalSide	水平停靠方向，默认值 Left

(续表)

属　　性	说　　明
VerticalOffset	距离浏览器的垂直边距，默认值 0px
VerticalSide	垂直停靠方向，默认值 Top
ScrollEffectDuration	滚动效果的延迟时间，单位为秒，默认值 0.1

下面通过【例 10-14】演示 AlwaysVisibleControlExtender 扩展控件的用法。

【例 10-14】演示 AlwaysVisibleControlExtender 扩展控件的用法。

(1) 启动 VS，打开 WebSite10 网站，创建名称为 AlwaysVisibleControlExtender.aspx 的网页。

(2) 在 AlwaysVisibleControlExtender.aspx 的【设计】视图中，放置一个 ScriptManager 和一个 Panel。在 Panel 中添加一个 ImageButton 控件。

(3) 单击 Panel 控件的【添加扩展程序】任务选项，绑定 AlwaysVisibleControlExtender 扩展控件。

(4) 该页面【源】视图中的部分代码如下。

```
    <%@ Register assembly="AjaxControlToolkit" namespace="AjaxControlToolkit" tagprefix="cc1"
%>
    • • •
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <asp:Panel ID="Panel1" runat="server"    Width="150px" height="120px" >
            <asp:ImageButton ID="ImageButton1" runat="server" ImageUrl="images/1.jpg" />
        </asp:Panel>
        <cc1:AlwaysVisibleControlExtender ID="avce" runat="server"
                TargetControlID="Panel1"
                VerticalOffset="10"
                HorizontalOffset="10"
                ScrollEffectDuration=".1" />
```

(5) 按 Ctrl＋F5 组合键运行，运行效果如图 10-23 所示，上下移动浏览器的滚动条，可以看到，图片始终保持在页面的左上角位置上。

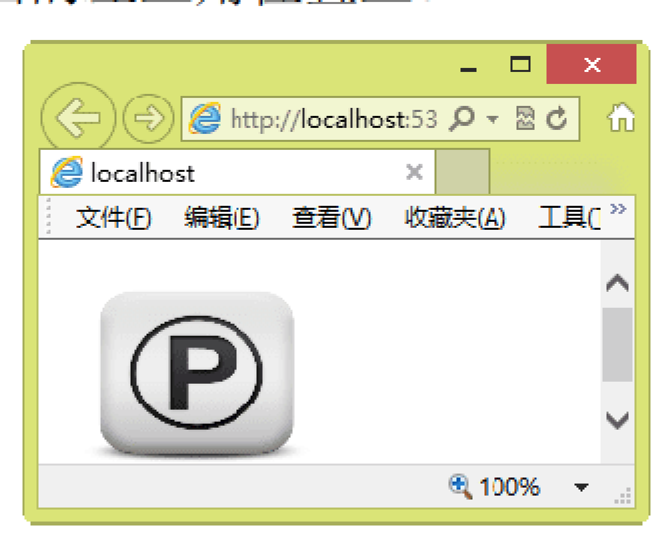


图 10-23　程序运行效果

ASP.NET AJAX Control Toolkit 是一个免费开源的工具包，包含了新的 Web 服务器控件，随着控件数量的逐渐增加，无疑会给 Web 应用程序开发带来更大方便。

10.7　本章小结

本章全面讲述了 ASP.NET AJAX 技术。首先简单介绍了 AJAX 的概念；然后介绍 ScriptManager、UpdatePanel、UpdateProgress、Timer 控件在 AJAX 技术中的使用；最后介绍了 ASP.NET AJAX Control Extenders 扩展控件的使用。通过本章的学习，读者可以全面的掌握 AJAX 技术在 ASP.NET 应用程序中的使用方法。

10.8　思考和练习

1. 简要介绍 AJAX 技术的原理。
2. 使用 AJAX 技术能带来什么优势?
3. ScriptManager 控件是 ASP.NET AJAX 的核心，它的作用是什么?
4. 利用 AJAX 工具控件包设计一个用户注册界面，注册的信息包括用户名、密码、生日等信息，要求：

(1) 所有输入信息不能为空；

(2) 用户在输入密码时要弹出提示对话框提示密码长度不能小于 6 位；

(3) 利用日期选择扩展控件让用户选择生日。

第11章　jQuery

目前流行的前端的语言是 JavaScript，它是一种动态类型、弱类型、基于原型的语言，内置支持类型。它的解释器被称为 JavaScript 引擎，为浏览器的一部分，广泛用于客户端的脚本语言。最早是在 HTML 网页上使用，用来给 HTML 网页增加动态功能。

虽然 JavaScript 的功能丰富且强大，但它仍然存在一些不足之处。例如，并不是所有的浏览器都以相同的方式解释 JavaScript。大部分 JavaScript 代码都可以在主流浏览器中运行，但是浏览器对代码的解释和表现出的行为存在一些细微的差别，所以很难编写能够在所有主流浏览器中表现出相同行为的代码。而且，JavaScript 还缺少能够使日常的 JavaScript 编码变得更加轻松的实用功能。例如，它提供了可以找到页面上特定元素的内置方法。例如，通过 getElementById 方法找到指定元素和通过 getElementsByTagName 方法找到特定 HTML 标记的全部元素。但是，它缺少找出应用到特定类的元素列表的功能，如 getElementsByClassName。

幸运的是，Internet 开发社区一直在致力于开发一种在后台使用 JavaScript 的框架。这种框架扩展了 JavaScript 的功能，同时也提供了非常丰富的功能集，用来帮助构建人员构建交互式的客户端 Web 页面。他们已经开发出了许多 JavaScript 库(其中大多数都是免费的)。目前主流的 JavaScript 库有：Prototype、Scriptaculous、Ext JS、Dojo、jQuery 等。其中，jQuery 最早由 John Resig 在 2006 年 1 月开发和发布，现在已经成长为一个备受欢迎的客户端框架。Microsoft 也注意到 jQuery 功能强大，并决定在自己的产品中附送这个框架。最初，jQuery 随 Microsoft ASP.NET MVC 框架一起提供，现在也包含在 Visual Studio 2012 中。

本章的学习目标：

- jQuery 简介。
- NuGet 的概念和用法。
- 如何使用 jQuery 增强页面，包括添加丰富的视觉效果和动画。
- 如何使用 jQuery 扩展 ASP.NET 验证框架。

11.1　jQuery 概述

jQuery 库简化了访问 Web 页面元素的方法、帮助处理客户端事件、提供视觉效果(如动画)支持，以及更加便于在应用程序中使用 Ajax。获得 jQuery 库并添加到 Web 站点上有以下几种方式。

首先，可以从 jQuery 的官方网站 http://jquery.com 上下载 jQuery 的最新版本。该网站

不但提供了可以下载的文件，还提供了文档、FAQ、教程和其他有助于更好地利用 jQuery 的信息。

第二种方式是使用 ASP.NET Web 窗体站点模板创建新站点，因为它已经包含一个带有必要 jQuery 文件的 Scripts 文件夹。

在站点中添加 jQuery 的第三种方式是使用 Content Delivery Network(CDN)。

最后一种方式是使用 NuGet，这是 Visual Studio(包括 Express 版本)附带的库包管理器。下面讨论 NuGet。

11.1.1　NuGet 简介

NuGet 是一个开源的库包管理器，它是 Visual Studio 的一个扩展，非常便于在 Visual Studio 项目和网站中添加、删除和更新外部库。它最初由 Microsoft 开发，但现在是一个开源项目，接受开发社区的支持。尽管在 VS 中 NuGet 用于网站，但 NuGet 不仅仅用于 ASP.NET，还可以用于用构建的所有类型的应用程序。

要在 ASP.NET 网站中添加一个开源库，如 jQuery，通常需要执行如下步骤。

(1) 找到库的网站。

(2) 找到该库的最新稳定版本的下载链接并下载它。

(3) 使用 Windows 资源管理器开启下载的文件，对它解压缩，添加到项目中，也可以为它创建一个专用的文件夹。

(4) 也可以在项目中添加一个对库的引用。

(5) 也可以通过 Web.config 文件或其他代码文件配置该库。

使用 NuGet 可以极大地减少把库添加到项目中所需要的步骤。例如，只需要单击 5 次，就可以添加 jQuery 库。除了添加包之外，把库添加到网站中后，NuGet 还可以方便地把它们更新为最新版本，也可以不留痕迹地删除包。

除了作为 VS 的扩展之外，NuGet 也有自己的官方网站 http://nuget.org。在这里可以浏览包的类别，查找安装包的命令，阅读详细文档等。

使用 NuGet 管理库有两个选项：使用【管理 NuGet 程序包】对话框和【程序包管理器控制台】。

1.【管理 NuGet 程序包】对话框

【管理 NuGet 程序包】对话框可以在线搜索包，管理已安装的包。要访问这个对话框，可以在【解决方案资源管理器】中右击网站，选择【管理 NuGet 程序包】，也可以选择【工具】|【库程序包管理器】|【管理解决方案的 NuGet 程序包】，打开【管理 NuGet 程序包】对话框，如图 11-1 所示。

在对话框的左边，可以选择 4 种不同类型的包列表：

- 【已安装的包】：这一项显示了已安装到网站中的包列表。使用这个选项可以确定已安装的包的版本、卸载包，或者把相同的包安装到同一个解决方案的另一个网站或项目中。

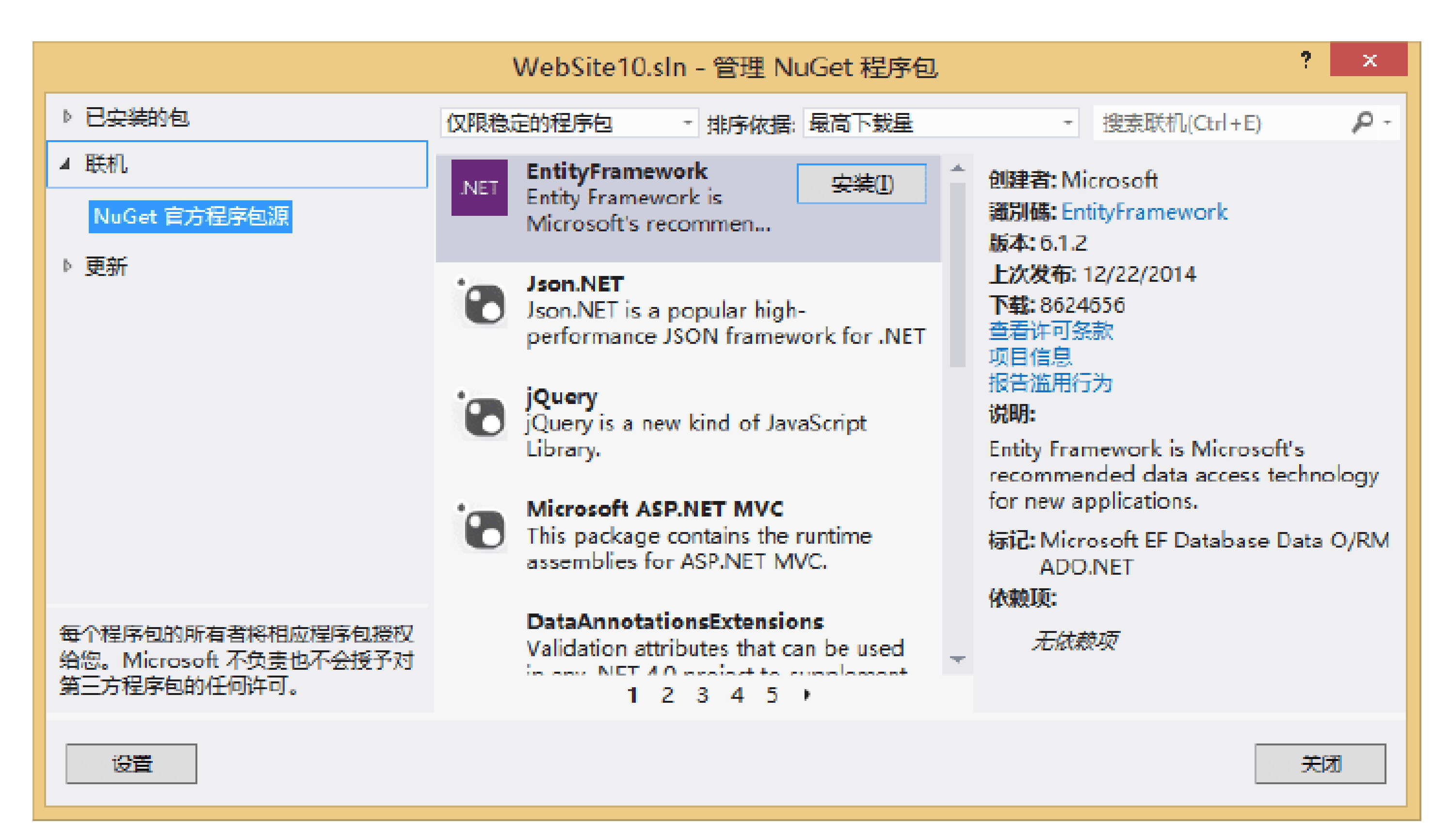


图 11-1　【管理 NuGet 程序包】对话框

- 【联机】：这一项允许使用官方的 NuGet 包源在线搜索包，或者可能已添加的其他种子(这可以使用【工具】|【选项】|【包管理器】|【程序包源】来实现)。默认情况下，该列表先显示最流行的包(用下载次数来衡量)，但可以使用顶部的下拉列表来改变顺序。在【排序依据】下拉列表的左边，可以选择只显示稳定的包，或者还包含包的预发布版本。在右上角，可以输入包的名称(或名称的一部分)来搜索包。一旦找到要添加到网站中的包，就选中它，单击【安装】按钮，【NuGet 包管理器】就会处理安装过程。
- 【更新】：这一项显示了自从把包安装到网站后发布了新版本的包列表。因为更新版本可能使已有的网站中断，所以它们不会自动安装，而需要手动选择要更新的包。
- 在 NuGet 网站的文档部分 http://bit.ly/q2PiyM，可以找到关于【NuGet 包管理器】对话框的完整文档。

除了使用这个对话框之外，还可以使用【程序包管理器控制台】管理包。

2. 【程序包管理器控制台】的使用

【程序包管理器控制台】允许在命令行界面上管理包。这不像使用对话框那样友好，但一旦喜欢使用 NuGet，就会发现这个控制台非常有用。使用该控制台，可以访问与对话框相同的功能，还能使用更多的功能。图 11-2 显示了【程序包管理器控制台】，选择【工具】|【库程序包管理器】|【程序包管理器控制台】就可以从主菜单中访问它。

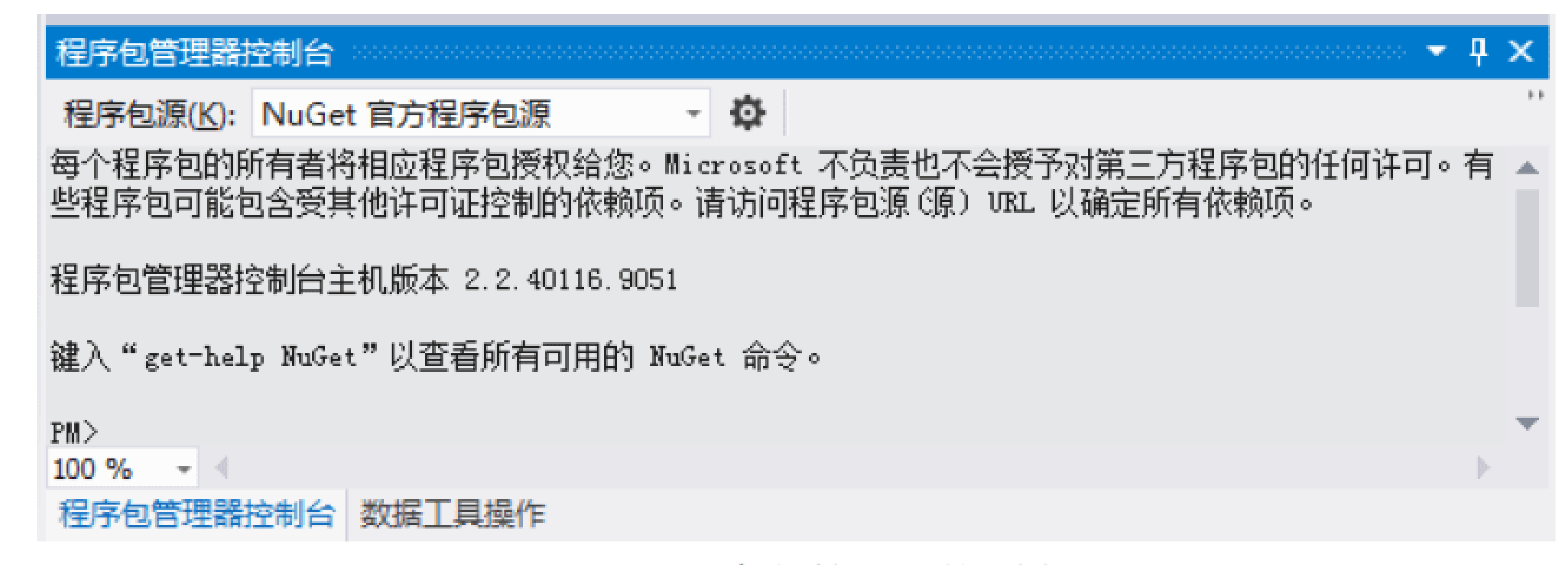


图 11-2　【程序包管理器控制台】

这个命令行窗口支持的许多命令都有助于管理包。为了获得主要的可用命令列表，可以输入 Get- Help NuGet，按下回车键，就会显示可以在控制台上运行的命令列表。这个控制台也有智能提示功能，按下 Tab 键，就可以帮助用户完成命令的输入。例如，Help 命令，只要输入 Get，再按下 Tab 键，就可以列出以 Get 开头的所有命令。接着可以从该列表中选择命令，再次按下 Tab 键，最后按下回车键，就可以完成该命令的输入。因为许多命令都以 Get 开头，所以输入 Get-H，再按下 Tab 键，就比较快，或者自己输入完整的命令。但是，对于很长、比较独特的命令名称而言，使用 Tab 键可以节省许多时间。

也许最常用的命令是 Get-Package 和 Install-Package。使用 Get-Package 可以列出当前在站点上安装的包。要查看该列表，只需要输入命令，再按下回车键。如果现在就尝试输入，列表就是空的，因为目前还没有安装任何包。使用 Get-Package-ListAvailable，就可以看到所有可用包的完整列表。因为有数千个这样的包(这个列表还在增加)，所以可以使用-Filter 过滤列表，代码如下。

```
Get-Package -ListAvailable -Filter jQuery
```

这列出了名称中有 jQuery 的所有包。注意这个过滤不区分大小写。要找到还未正式发布的包的预发布版本，可以使用-PreRelease 选项，代码如下。

```
Get-Package -ListAvailable -Filter jQuery -PreRelease
```

一旦找到需要安装的包，就使用 Install-Package 命令安装它。需要给该命令提供包的名称作为参数。例如，要把 jQuery 安装到网站中，可以执行如下命令。

```
Install-Package jQuery
```

按下 Enter 键后，【程序包管理器控制台】就会下载该包，把它安装到网站上，这与使用【管理 NuGet 程序包】对话框的效果一样。

【例 11-1】练习使用管理 NuGet 程序包对话框在网站上安装 jQuery 库。

这个练习将介绍如何使用管理 NuGet 程序包对话框查找并安装最新的 jQuery 包。使用这个对话框是最直观、最简单的方式，所以这是使用 NuGet 的最佳起点。在本书的后面部分，还将使用控制台窗口安装包。

(1) 首先打开【管理 NuGet 程序包】对话框，如图 11-1 所示。

(2) 如果还未选择，就单击左边列表中的【联机】项。

(3) 找到 jQuery 项，如果希望安装的包没出现在联机列表中，就使用对话框右上角的搜索框。

(4) 单击列表中的 jQuery 库包。确保只选中 jQuery 库，而不选中 jQuery UI(Combined Library，即：合并的库)或 jQuery Validation 包，因为它们用于不同的目的。

(5) 单击【安装】按钮，弹出一个对话框，其中列出了安装包所执行的动作。这个对话框消失后，jQuery 安装完毕，【解决方案资源管理器】中出现一个 Script 文件夹，单击【关闭】按钮，退出【管理 NuGet 程序包】对话框。

注意：

在安装过程中，NuGet 在 Scripts 文件夹中添加了 3 个与 jQuery 有关的 JavaScript 文件。这个文件放在网站的根上，并创建一个 packages 文件夹来存储已下载包的本地副本。

Scripts 包中包含 3 个.js 文件。其中，jquery-2.1.3.js 文件包含未压缩格式的核心 jQuery 代码，这意味着可以读取其代码。在需要查看库执行了什么代码时，将这些代码用于调试。jquery-2.1.3. intellisense.js 文件包含 jQuery 库中用于 IntelliSense 列表和文档弹出窗口的代码的文档。jquery-2.1.3.min.js 文件是压缩格式的核心 jQuery 库，这个文件用于网站，因为它最小，因此下载起来最快。

把 jQuery 库添加到网站中后，下一步就是确定在哪里使用它。因为 jQuery 库增加了网页的大小，所以应慎重选择是否在站点中包含它。

11.1.2 设置引用 jQuery 的位置

要在 Web 站点中引用 jQuery，有以下几种方法。

- 只在需要 jQuery 的网页或者用户控件中添加对 jQuery 库的引用。
- 在 Web 站点的母版页中添加对 jQuery 库的引用，这样所有的页面都可以使用该 jQuery 库。

两种方法各有优缺点。只在需要 jQuery 库的页面上添加对它的引用，可以减小页面大小。当用户浏览没有使用 jQuery 的页面时，就不需要下载 jQuery 库文件，节省了一些带宽。而当他们访问了引用库文件的页面以后，浏览器就会缓存库文件的一个副本，以后访问页面时，就不需要再次下载这些文件。

在 Web 站点的母版页中添加对 jQuery 库的引用十分方便，因为所有基于该母版页创建的页面都会自动获得对 jQuery 功能的访问权。但是，这会对 Web 站点第一个页面的性能造成一些冲击，因为需要从服务器上下载库文件。

除了添加 jQuery 文件的位置以外，还有一些关于如何包含库文件的选项。

11.1.3 包含 jQuery 库的方式

因为 jQuery 库由一个使用 JavaScript 代码编写的文件组成，所以可以使用标准的<script>语法在页面、用户控件或母版页中嵌入对 jQuery 库的引用。

```
<script src="FileName.ext" type="text/javascript"></script>
```

必须使用一个独立的结束</script>标记，因为如果使用自结束标记，一些浏览器无法正常运行代码。

【NuGet 包管理器】默认为 JavaScript 文件(如 jQuery)，使用在 Web 站点根目录下创建 Scripts 文件夹，所以对 jQuery 库(jquery-2.1.3.min.js)的引用如下。

```
<script src="/Scripts/jquery-2.1.3.min.js" type="text/javascript"></script>
```

也可以将引用嵌入到母版页的 ScriptManager 控件中。ScriptManager 控件有一个

<Scripts>子元素，可以用来注册 JavaScript 文件，该文件将添加到浏览器的最后一个页面上。在 ScriptManager 中注册的 JavaScript 文件的最简形式如下。

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
<Scripts>
<asp:ScriptReference Path="~/Scripts/jquery-2.1.3.min.js" />
</Scripts>
</asp:ScriptManager>
```

另外一种方法是使用 Microsoft 的内容传送网络(Content Delivery Network，简称 CDN)或 Google Code 引用 jQuery 库的在线版本。

使用外部库的在线版本的优势在于可以提升服务器的性能并降低带宽。因为站点的访问者很可能已经在访问另外一个站点时下载了共享脚本，所以当用户访问站点时，就不需要再次下载这些文件了。

在下面的【例 11-2】中，将向站点的母版页添加 jQuery 库。在设置好 jQuery 库后，本章其他部分将会讨论 jQuery 的工作原理，以及如何在站点中利用 jQuery。

【例 11-2】将向母版页添加 jQuery 库，从而使站点内的所有页面都可以访问它。

(1) 新建 MasterPages 文件夹，然后新建 Frontend.master 文件，添加 ScriptManager 控件，切换到【源】视图，添加 jquery 引用，代码如下。

```
<asp:ScriptManager ID="ScriptManager1" runat="server" EnablePageMethods="true">
<Scripts>
<asp:ScriptReference Path="~/Scripts/jquery-2.1.3.min.js" />
</Scripts>
</asp:ScriptManager>
```

(2) 保存并关闭母版页。

(3) 创建需要引用 jquery 的 Web 页面，将该页面命名为 jQuery.aspx，并将它的 Title 设置为 jQuery Demo。

(4) 在【源】视图中打开页面，向 ContentPlaceHolder1 的 Content2 代码块中添加如下代码。

```
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
<input id="Button1" type="button" value="button" />
<script type="text/javascript">
    $(document).ready(function () {
        $('# Content2').css('background-color', 'green')
        $('#Button1').click(function () {
            $('# Content2').css('background-color', 'red')
            .animate({ width: '100px', height: '800px' })
        });
    });
</script>
</asp:Content>
```

和其他许多编程语言一样，JavaScript(以及 jQuery)是区分大小写的，对缺少引号、大

括号和小括号十分敏感，所以一定要注意。

(5) 将改动保存到页面中，然后按下 Ctrl+F5 组合键，在浏览器中打开该页面。注意，Content2 元素的背景色变成了绿色。单击按钮，注意背景色变成了红色，而 Content2t 元素的大小也发生了改变，最终变成了 100 像素宽，800 像素高。

注意：

如果出现了错误，或者没有看到动画，那么一定要确保正确地在页面中添加了指向 jQuery 的链接。另外，检查代码中是否存在输入错误。

本例中，首先添加了对 jQuery 库的引用。

```
<asp:ScriptManager ID="ScriptManager1" runat="server" EnablePageMethods="true">
<Scripts>
<asp:ScriptReference Path="~/Scripts/jquery-2.1.3.min.js" />
</Scripts>
</asp:ScriptManager>
```

这段代码告诉脚本管理器包含一个指向 jQuery 库的 script 元素。如果在浏览器中查看页面的 HTML 源代码，那么应该看到如下 script 元素。

```
<script src="../Scripts/jquery-2.1.3.min.js" type="text/javascript"></script>
```

这行代码告诉浏览器从 Scripts 文件夹中下载 jquery-2.1.3.min.js 文件，使页面可以访问 jQuery 库提供的全部功能。

接下来需要查看的是 jQuery 演示页面的代码。首先，添加了一个标准的<script>块，其中可以包含 JavaScript。在这个块中，添加了一些在浏览器加载页面完成后触发的 jQuery 代码。页面就绪后，会执行起始大括号({)和结束大括号(})之间的代码。

```
<script type="text/javascript">
$(document).ready(function() {
// Remainder of the code skipped
});
</script>
```

因为 jQuery 代码与页面上的元素进行交互，所以经常需要等待整个页面加载完成，然后才能通过编程操作元素。添加这样的 jQuery 代码是将代码执行延迟到页面就绪后的一种标准做法。例如，本章后面访问“文档就绪函数”的便捷方式$(document).ready。

页面就绪后执行的代码包括两部分。代码的第一行将 Content2 <section>的背景色设置为绿色。

```
$('# Content2').css('background-color', 'green')
```

这行代码获取对 Content2 元素的引用，然后调用 css 方法将背景色修改为绿色。document.getElementById 方法通过 id 获取对页面中某个元素的引用。在本例中，$('# Content2')是 jQuery 中与之等效的方法。

代码的第二部分为添加到页面的 HTML 按钮建立一个单击处理程序，类似于 onclick。在该单击处理程序中，可以看到一些代码将 Content2<section>元素的背景色修改为红色，还有一些代码使用流体动画改变其宽和高。

```
$('#Button1').click(function() {
$('# Content2').css('background-color', 'red')
.animate({ width: '100px', height: '800px' })
});
```

在浏览器中单击按钮时，Content2 的背景色就会变成红色，而它的宽和高将会分别变成 100 像素和 800 像素。

11.2　jQuery 语法基础

要理解和使用 jQuery，需要掌握一些基础知识。首先，需要更深入地理解 jQuery 的核心功能，包括前面看到的$函数及其 ready 方法。通过使用 ready 方法，当浏览器完成页面的加载后，可以执行一些代码。接下来，需要学习 jQuery 的 selector 和 filter 语法，这样就可以通过自己指定的条件在页面中查找元素。当获得指向页面中一个或多个元素的引用后，就可以对它们应用多种方法，如前面提到的 css 方法。还需要知道关于 jQuery 事件的一些知识，因为它们允许向 HTML 元素可能触发的事件(例如 click、mouseover 等)附加行为。接下来简单介绍 jQuery 的语法。

11.2.1　jQuery 核心知识

由于大部分 jQuery 代码将在浏览器完成页面加载后执行，因此，等到页面完成 DOM 加载后再执行代码十分重要。DOM(Document Object Model，即：文档对象模型)是 Web 页面的一种分层表示，包含所有 HTML 元素、脚本文件、CSS、图像等的一个树形结构。DOM 总是与用户在浏览器中看到的页面保持一致，所以如果借助编程修改 DOM(如使用 jQuery 代码)，那么这种修改将在浏览器显示的页面上反映出来。如果过早执行 jQuery 代码(如在页面的最顶端)，那么 DOM 就可能还没有加载脚本中引用的全部元素，这会产生错误。因此，使用 jQuery 中的 ready 函数，将代码的执行推迟到 DOM 就绪。示例代码如下。

```
$(document).ready(function() {
// Code added here is executed when the DOM is ready.
});
```

这样，当页面准备就绪，可以执行 DOM 操作时，就执行添加到起始和结束大括号之间的全部代码。jQuery 也提供了 ready 函数的一个快捷方式，便于编写当 DOM 就绪后触发的代码。示例代码如下。

```
$(function() {
// Code added here is executed when the DOM is ready.
```

```
});
```

因为 jQuery 代码通常针对每个页面，所以应该只在需要 jQuery 代码的页面结尾添加代码。为了简化这项任务，可以在母版页中添加一个 ContentPlaceHolder，使用这个母版页的页面就有了一个地方来编写 jQuery 代码。例如，前面的示例在页面中使用 Content 2<section>元素和按钮的代码。

11.2.2　使用 jQuery 进行选择

在 jQuery 中，可以使用美元符号($)在页面中查找元素。找到并返回的元素称为匹配集(matched set)。$方法的基本语法如下。

```
$('Selector Here')
```

在引号之间，输入一个或多个选择器。$方法返回 0 个或多个元素，然后可以使用多种 jQuery 方法操作这些元素。例如，要向所有 h2 元素应用 CSS，可以使用 CSS 方法：

```
$('h2').css('padding-bottom', '10px');
```

这行代码使页面中的所有二级标题具有 10 像素高的底内边距。许多 jQuery 方法的优点在于，除了应用某些设计或行为，它们会再次返回匹配集，这样可以对相同的匹配集调用其他方法。这个概念称为链接或 fluent 编程，即使用一个方法的结果作为另一个方法的输入，从而产生一个效果链。示例代码如下。

```
$('h2').css('font-size', '40px').fadeOut(5000);
```

下面的代码首先修改了页面中全部二级标题的字体，然后在 5s 内让它们淡出，直至不可见。

1. 基本选择器

jQuery 选择器能够在页面的文档对象模型中找到一个或多个元素，以便向它们应用各种类型的 jQuery 方法。前面章节中介绍的 CSS 选择器，在 jQuery 中完全可以使用。

(1) 通用选择器

与对应的 CSS 选择器一样，通用选择器会匹配页面中的全部元素。例如，要将页面中每个元素的 style 的 font-family 属性设置为 Arial，代码如下。

```
$('*').css('font-family', 'Arial');
```

(2) 元素选择器

这个选择器获得与特定的标记名相匹配的 0 个或多个元素的引用。例如，下面的代码将所有二级标题的文本颜色设置为蓝色。

```
$('h2').css('color', 'blue');
```

(3) ID 选择器

与对应的 CSS 选择器一样，这个选择器通过 id 查找和获取元素。例如，要为名为 Button1

的按钮设置 CSS 类，代码如下。

```
$('#Button1').addClass('NewClassName');
```

当这行代码使用 addClass 方法设置 CSS 类时，将会遵循标准的 CSS 规则。这意味着要想使这行代码生效，并修改按钮的外观，需要通过外部 CSS 文件或者嵌入式样式表将 NewClassName 类提供给页面。

还有，把不存在的 CSS 类添加到 CSS 文件中。这是“标记”元素的方便方式，以便日后通过类选择器再次选择它们。

(4) 类选择器

类选择器获得与特定的类名相匹配的 0 个或多个元素的引用。例如，下面的 HTML 代码段。

```
<h1 class="Highlight">Heading 1</h1>
<h2>Heading 2</h2>
<p class="Highlight">First paragraph</p>
<p>Second paragraph</p>
```

注意，4 个元素中有两个元素都有一个名为 Highlight 的 CSS 类。下面的 jQuery 代码将第一个标题和第一个段落的背景色修改为红色，而保持其他元素不变。

```
$('.Highlight').css('background-color', 'red');
```

(5) 分组和合并选择器

与 CSS 一样，可以分组和合并选择器。下面的分组选择器修改页面中全部 h1 和 h2 元素的文本颜色。

```
$('h1, h2').css('color', 'orange');
```

使用合并选择器，可以找出其他一些元素包含的特定元素。例如，下面的 jQuery 只修改 Content2 元素中包含的段落，而保持其他所有段落不变。

```
$('#Content2 p').css('border', '1px solid red');
```

下面的【例 11-3】演示了 jQuery 选择器的使用。

【例 11-3】首先向主母版页添加一个额外的 ContentPlaceHolder 控件，以便于向页面添加客户端 jQuery 代码，然后编写一些 jQuery 来试验各种选择器。

(1) 打开 Frontend.master 文件，切换到【源】视图中。

(2) 在页面底部附近，结束标记</form>之前，从工具箱中拖动出一个 ContentPlaceHolder 并放到这里。将其 ID 设置为 cpClientScript。自动生成的代码如下。

```
<footer>Footer Goes Here</footer>
</div>
<asp:ContentPlaceHolder ID="cpClientScript" runat="server">
</asp:ContentPlaceHolder>
</form>
```

(3) 保存并关闭母版页。

(4) 在 Demos 文件夹中创建一个新的演示页面，并将其命名为 BasicSelectors.aspx。同样，基于模板创建这个页面，并给它添加标题。将页面切换到【设计】视图中，在底部找到 cpClientScript 占位符，打开它的任务面板，然后选择【新建自定义内容】命令。

(5) 切换到【源】视图，将下面的 HTML 代码添加到 cpMainContent 占位符中。

```
<h1>Basic Selectors</h1>
<div class="SampleClass">I am a div.</div>
```

(6) 在第(4)步创建的 cpClientScript 占位符中添加如下 jQuery 代码。

```
    <asp:Content ID="Content3" runat="server" ContentPlaceHolderID="cpClientScript">
    <script type="text/javascript">
    $(function()
    {
    $('*').css('color', 'Green');

    $('#Sidebar').css('border-bottom', '2px solid red');

    $('h1').bind('click', function () { alert('Hello World') });
    $('.SampleClass').addClass('PleaseWait').hide(5000);
// Class
    $('footer, header').slideUp('slow').slideDown('slow');
    $('#Sidebar img').fadeTo(5000, 0.1);

    });
    </script>
    </asp:Content>
```

这些代码演示了 6 个基本选择器的使用。

(7) 保存并在浏览器中浏览页面，所有的文本都是绿色的，边栏有一个额外的底边框，还可以看到 Please Wait 动画的图标和文本显示然后消失，页眉和页脚缓慢消失然后重新显示，以及在 5 秒钟的时间段内，边栏的横幅变为几乎透明。如果单击基本选择器标题，将会弹出一个小窗口显示 Hello World。

2. 基本筛选器

在 jQuery 中，可以使用筛选器找到特定的元素，如第一个元素、最后一个元素、所有奇数行元素、所有偶数行元素、所有的标题或者特定位置的项。下面的【例 11-4】演示了最常用的基本筛选器，并给出了一个查找特定元素的示例。

【例 11-4】创建一个页面，演示查找特定元素。

(1) 基于定制模板创建一个新页面，命名为 jQueryDemos.aspx。给页面指定标题，然后，在 cpMainContent 占位符中添加下面的 HTML 代码段。

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="cpMainContent" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
```

```
        <h1 title="First Header">First Header</h1>
    <table id="DemoTable">
    <tr><td>Row 1 Cell 1</td><td>Row 1 Cell 2</td></tr>
    <tr><td>Row 2 Cell 1</td><td>Row 2 Cell 2</td></tr>
    <tr><td>Row 3 Cell 1</td><td>Row 3 Cell 2</td></tr>
    <tr><td>Row 4 Cell 1</td><td>Row 4 Cell 2</td></tr>
    <tr><td>Row 5 Cell 1</td><td>Row 5 Cell 2</td></tr>
    </table>
    <h2>Second <span style="font-style: italic; font-weight: bold;">
    Header</span></h2>
    <input id="Button1" type="button" value="button" />
    <input id="Text1" type="text" />
    <input id="Checkbox1" type="checkbox" />
    <input id="Checkbox2" type="checkbox" />
    </asp:Content>
```

(2) 在母版文件中添加一个 ID 为 cpClientScript 的 Content 块。返回内容页，输入下面的代码，在 cpClientScript 的 Content 块中添加如下代码。

```
    <asp:Content ID="Content2" runat="server" contentplaceholderid="cpClientScript">
        <script type="text/javascript">
            $(function () {
                //Examples go here
            });
    </script>
    </asp:Content>
```

(3) 在//Examples go here 处添加以下代码。

```
    $('#DemoTable').css('background-color', 'green');
```

(4) 保存修改，按下 F5 键，在浏览器中打开该页面，表中单元格的背景色将变为绿色，如图 11-3 所示。



图 11-3　运行效果

如表 11-1 所示中，列出了 jQuery 的基本筛选器。记住，要试验每个示例，可以使用表中给出的代码示例替换//Examples go here 一行，然后保存页面，并在浏览器中加载页面，查看代码的效果。

表 11-1　jQuery 的基本筛选器

筛　选　器	用　　途
:first :last	用于选择匹配集中的第一个和最后一个项。下面的示例将表的第一行和最后一行的背景色设置为红色。 $('#DemoTable tr:first').css('background-color', 'red'); $('#DemoTable tr:last').css('background-color', 'red'); 首先，使用#DemoTable 找到表。然后使用 tr 找到表的全部行。最后，使用:first 和:last 筛选器找到第一行和最后一行

(续表)

筛　选　器	用　　途
:odd :even	用于选择匹配集中的奇数行或者偶数行。下面的示例将表的奇数行的背景色修改为红色。因为计数是从零开始的，所以实际上第 2 行和第 4 行的背景色发生了改变(因为它们的索引分别为 1 和 3)。 $('#DemoTable tr:odd').css('background-color', 'red');

:eq(index)
:lt(index)
:gt(index)
	按照索引匹配元素。:eq(equals)根据索引返回一个元素，而:lt(less than)和:gt(greater than)则分别返回小于或者大于给定索引的项。示例如下所示。
	// Changes the color in the first row (with an index of 0)
	$('#DemoTable tr:eq(0)').css('color', 'green');
	
	// Changes the last two rows (rows 1, 2 and 3,
	// with an index of 0, 1 and 2 respectively, are skipped.)
	$('#DemoTable tr:gt(2)').css('color', 'green');
	
	// Changes the text color of the first two rows to green.
	$('#DemoTable tr:lt(2)').css('color', 'green');

:header	找到页面中的全部标题(从 h1 到 h6)。示例如下所示。 $(':header').css('color', 'green');

要查看所有基本筛选器的完整列表，可以阅读 jQuery 文档，网址为 http://api.jquery.com/category/ selectors/。

3. 高级筛选器

除了刚才看到的基本筛选器以外，jQuery 还支持其他很多筛选器，用来根据项包含的文本、是否可见以及它们包含的任意属性获取项。另外，还有一些筛选器可以获得窗体元素(例如按钮、复选框、单选按钮等)，以及大量可以选择子元素、父元素、兄弟元素和后代元素的选择器。如表 11-2 所示列出了最常用的筛选器。

表 11-2

筛　选　器	用　　途
:contains(text)	通过包含的文本匹配元素。示例如下。 $('td:contains("Row 3")').css('color', 'green'); 如果省略 td，那么整个表都会变成绿色。这是因为表本身也会匹配(它的一个子表包含文本 Row 3)，所以颜色将应用到整个表上，使每个单元格的文本变为绿色。注意文本字符串使用了双引号，以避免过早关闭选择器中的单引号
:has(element)	匹配至少包含一个给出元素的元素。示例如下。 $(':header:has("span")').css('color', 'green'); 这只匹配 h2，因为它是标题(:header)，并包含<span>元素(has("span"))

(续表)

<table>
<tr><th>筛　选　器</th><th>用　　途</th></tr>
<tr><td>[attribute]</td><td>基于给定属性匹配元素。示例如下。
// Matches the button and the text box as both
// have a type attribute
// match other elements with a type attribute
$('[type]').css('color', 'green');
为了只选择带 type 特性的输入控件，可以使用如下代码。
$('input[type]').css('color', 'green');
需要在文本框中输入一些文本来查看绿色的字体</td></tr>
<tr><td>[attribute=value]</td><td>基于一个属性和该属性的值匹配元素。示例如下。
// Matches just the text box
$('[type=text]').css('color', 'green');</td></tr>
<tr><td>:input
:text
:password
:radio
:checkbox
:submit
:image
:reset
:button
:hidden
:file</td><td>这些选择器可以匹配特定的客户端 HTML 表单元素。例如，可以使用分组选择器把查找按钮和文本框的代码段重写如下。
$(':button, :text').css('color', 'green');
可以使用这些筛选器来实现一些奇妙的效果。例如，要编写一些功能来选中表单中的全部复选框，可以使用下面的代码：
$(':checkbox').attr('checked', true);
要想取消选择全部复选框，可以传递 false 作为 attr 方法的第二个参数</td></tr>
</table>

11.3　通过 jQuery 修改 DOM

有了匹配集之后，就需要对它执行一些操作。例如，可能对匹配集中的项应用 CSS 类或者样式。或者，可能向它们附加一些行为，如添加一个 click 处理程序，项被单击时就可以触发一些代码。接下来将介绍如何操作各种 CSS 方法，学习如何建立事件处理程序。

11.3.1　css 方法的使用

jQuery 以几种不同的方式支持 CSS。首先，可以使用 css 方法来检索特定的 CSS 值，如某个项的颜色，以及设置一组元素的一个或多个 CSS 属性。其次，addClass、removeClass、toggleClass 和 hasClass 等方法可以修改或检查对元素应用的 CSS 类。再次，有几种方法可以修改元素的尺寸和位置，这里只讨论最常用的方法。这里的讲解使用了与前面相同的 HTML 代码段，所以很容易在浏览器中测试这些代码。

1. css(name,value)

这个方法可以设置某个匹配元素上的特定 CSS 属性。其中，name 参数引用了一个 CSS 属性，如 border、color 等，value 定义了要应用的样式。例如，下面的代码修改了 h3 元素的背景色。

```
$('h3').css('background-color', 'green');
```

2. css(name)

这个方法基于传递给它的属性检索特定的 CSS 值。例如，以下代码将提示'italic'，因为这是一级标题中<span>元素的 font-style。

```
alert($('h1 span').css('font-style'));
```

可以在 jQuery 脚本中使用这个值。例如，可以用来在斜体和普通字体之间切换 font-style，或者将多个元素设置为相同的类型。

3. css(properties)

这个方法可以同时设置匹配元素的多个属性。下面的代码将表中所有单元格的颜色修改为蓝色，将字体修改为 Verdana，并将内边距设置为 5px。数据用所谓的匿名对象来传递，其中完整的属性集包含在一对花括号({})之间，每个属性和属性值由冒号(:)分隔，而每对属性和属性值之间由逗号分隔。

```
$('#DemoTable td').css({'color' : 'blue', 'font-family' : 'Verdana',
'padding' : '5px'});
```

4. addClass、removeClass 和 toggleClass

addClass 和 removeClass 方法分别在元素中添加和删除类。和普通的 CSS 一样，使用这些方法，比使用 css(properties)方法进行内联 CSS 赋值更好，这样就更容易在一个集中的位置定义 CSS 类，易于维护和重用。例如，以下代码将一个类赋给 h1 元素。

```
$('h1').addClass('PleaseWait');
```

如果希望再次删除类，则可以调用 removeClass 方法，代码如下。

```
$('h1').removeClass('PleaseWait');
```

如果类还不存在，则 toggleClass 方法将分配一个类；否则，它将删除类。

所有这 3 个方法都允许传递多个类，各个类之间使用空格分隔。

5. attr(attributeName)

attr 方法可以读取和设置 HTML 元素上的特性值。Attr 方法的第一个版本，接收一个参数，对于 attributeName，应把特性名传递为字符串。下面的代码会显示 Second Header，即 h2 标题的 title 特性。

```
alert($('h2').attr('title'));
```

6. attr(attributeName,value)

Attr 方法的第二个版本，接收两个参数，用于修改特性的值，另外还需要给特性传送新值。例如，下面的代码选中页面中的所有复选框。

```
$(':checkbox').attr('checked', true);
```

结合使用这些 CSS 方法，能够在修改页面元素的外观上拥有极大的控制权。可以在此基础上，使用 jQuery 的丰富事件系统，通过代码向元素分配和删除各种处理程序。

11.3.2　事件及事件处理程序

前面章节中已经介绍了.NET 事件的应用，它们用于处理 Button 控件的 Click 事件或 Page 的 Load 事件。在这一点上，JavaScript 和 DOM 也不例外。例如，许多 HTML 元素(比如使用 input type="button"定义的按钮)都有一个 click 事件，在单击时触发。同样，它们还有 mouseover 和 mouseout 事件，当鼠标经过它们或者离开它们时触发。通常，在标记中直接定义事件的形式如下。

```
<input type="button" onclick="alert('你好');" value="确定" />
```

不要编写内联触发的代码(在本例中为 alert 函数)，最好将要执行的代码编写成 JavaScript 函数。下面的代码调用了一个虚构的 writeName 函数。

```
<input type="button" onclick="writeName ();" value="确定" />
```

jQuery 则更进一步，不仅允许将事件挂钩到单个元素上，还允许将事件挂钩到整个匹配集上。这种功能极为强大，因为只用几行代码，就可以将处理程序绑定到大量的元素上。例如，考虑具有许多行的一个表。为了使表格的外观美观一些，可以应用一种叫做“活动项跟踪”的技术，当鼠标移动到某个项上时，该项就改变颜色。如果不使用 jQuery，就需要对表的每一行编写 onmouseover 和 onmouseout 事件处理程序。这显然会显著增加页面最终的 HTML 代码量。而使用 jQuery 实现，代码如下。

```
$(function()
{
$('#DemoTable tr')
.bind('mouseover', function() { $(this).css('background-color', 'yellow') });
});
```

这些代码找出#DemoTable 元素中的全部表行，然后动态分配一个函数，当鼠标悬停在每一行上时，将会调用该函数。如果将鼠标悬停在行上，那么背景将会改变颜色。但是如果移走鼠标，那么新颜色将会保留下来。为了解决这个问题，可以使用 jQuery 的链接概念，即 jQuery 方法的结果会返回一个匹配集，所以可以对该结果应用其他函数。要将 mouseout 绑定到一个新函数，只需要对 bind 的第一次调用返回的值再次调用 bind，代码如下。

```
$('#DemoTable tr')
.bind('mouseover', function() { $(this).css('background-color', 'yellow') })
.bind('mouseout', function() { $(this).css('background-color', '') });
```

注意，在前一个示例中结束行的分号移动到了最后一行，这样，第二个 bind 就绑定到了前一次对 bind 的调用上。因为在这里代码放到了多行中，这些代码实际上等同于如下代码。

```
$('#DemoTable tr').bind('mouseover', ...). bind('mouseout', ...);
```

这些代码完成三项工作：首先，使用$('#DemoTable tr')找出表中的全部行。它在返回的匹配集中调用 bind，以便动态挂钩一些行为，当鼠标移动到某一行上时，就会触发这些行为。然后，对第一次调用 bind 返回的匹配集再次调用 bind，以便当鼠标从该行移走时重设置背景色。注意，代码中将颜色设置为一个空字符串(' ')，以便删除 CSS 背景属性，这样就可以再次看到原来的背景。

以下代码设置背景色。

```
$(this).css('background-color', 'yellow')
```

其中，this 关键字指的是应用该项的元素：在本例中就是表行。使用$(this)将得到 jQuery 匹配集(包含单个元素)，可以对其应用常规的 jQuery 方法，如 css 方法。如果不使用 jQuery，也可以对 this 元素执行标准的 JavaScript 方法，代码如下。

```
this.style.backgroundColor = 'yellow'
```

表行有一个 style 属性，可以用来通过编程方式修改 CSS 样式。

11.3.3　jQuery 的其他功能

1. each 方法

each 方法用于遍历一个集合。当需要对匹配集中的项应用某种行为，但是无法使用一个 jQuery 函数完成设置时，可以使用 each 方法。在使用时，需要把希望对每一项执行的函数作为参数传递给 each。下面的 each 示例通过遍历匹配集中的每一项，然后调用 alert，将每个表单元格的内容显示了出来。

```
$('#DemoTable td').each(function()
{
alert(this.innerHTML);
});
```

2. parent 方法和 prev 方法

parent 方法和 prev 方法用在 DOM 遍历中，允许在文档树中上下移动，找出低于、高于某一项或者与该项位于同一层次的元素。

prev 方法选择匹配元素的直系兄弟。其工作原理如下代码所示。

```
alert($('#DemoTable td:contains("Row 1 Cell 2") ').prev()[0].innerHTML);
```

这个警报的结果就是 Row 1 Cell 1。$选择器首先根据其包含的文本选择了第一行第二列的表单元格。之后 prev 方法返回它的第一个兄弟：其左侧的单元格。因为匹配集是一个集合，即使它只包含一项也是如此，所以仍然需要使用索引器(使用[0])来引用第一项。然后表单元格将显示一个.innerHTML 属性，它将返回该单元格的内容。

最后，parent 的代码如下，请参考。

```
alert($('#DemoTable td:contains("Row 1 Cell 2") ').parent()[0].innerHTML);
```

运行代码，结果如图 11-4 所示。

如果没有看到完全相同的 HTML，那么要确保以 alert 开头的一行是文档就绪函数中唯一的一行，因为其他示例可能影响到表的 HTML。parent 函数指向围绕匹配的表单元格的<tr>。显示 innerHTML 将返回该行包含的两个单元格的 HTML。

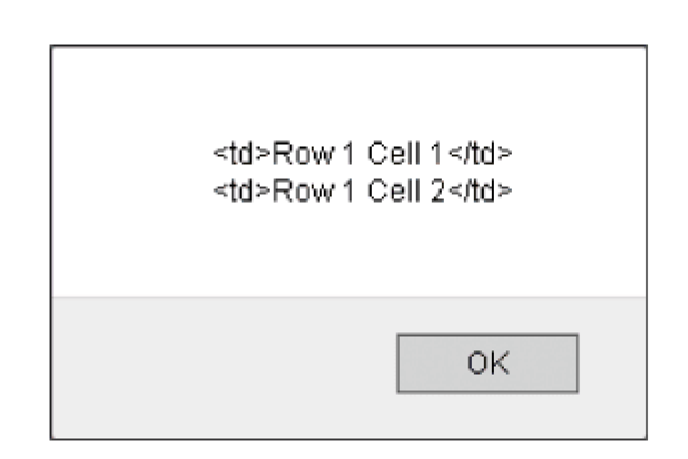


图 11-4　运行结果

11.3.4　使用 jQuery 时的常见错误

使用 jQuery 时，必须正确地书写大写字母、小括号、中括号、逗号等。当发现 jQuery 代码无法运行或者表现出异常行为时，检查代码是否存在下面列出的问题。

1. ID 选择器无效

当发生ID选择器无效时，很可能是忘记了在它的前面包含一个#符号。例如，$('DemoTable')，程序运行过程中，并不会选择id为DemoTable的表，而$('#DemoTable')才会选择该表。

ID 选择器无效的另外一种情况是大小写有误。ID 选择器是区分大小写的，所以，$('#DemoTable')与$('#demotable')是不同的。

2. 即使使用#符号，ID 选择器依然无效

出现这种情况时，很可能是客户端元素没有要查找的 id。是不是 ASP.NET 运行库修改了客户端 id，使用其父元素的 id 给它添加了前缀？如果是这样，尝试将相关控件的 ClientIDMode 设置为 Static，这样 id 的可预测性更好。另外，确保代码在文档准备函数中执行，因为很可能要查找的元素还不可用。

3. 没有代码可运行

检查小括号、大括号和引号。每种符号的开始符号和结束符号的数量必须相等。

前面的代码大都用于查找页面中的元素，通过 CSS 或执行 JavaScript 来改变其外观。但 jQuery 带有许多功能，还可以使页面中的元素具备动画效果。下一节将介绍这些效果，然后用一个示例说明其工作原理。

11.4 使用 jQuery 的效果

前面介绍了如何使用 slideUp 和 slideDown 实现元素淡入淡出效果，这是 jQuery 提供的诸多效果和动画方法中的两个。如表 11-3 所示的是最常用的一些 jQuery 方法。同样，可以使用 jQueryDemos.aspx 页面来试验这些效果。

表 11-3 常用的 jQuery 效果方法

方 法 名	用 途
show() hide()	通过递减 height、width 和 opacity(使它们变为透明)隐藏或者显示匹配元素。两种方法都允许定义一个固定的速度(慢、中、快)或者一个定义动画持续时间(单位为毫秒)的数字。示例如下。 $('h1').hide(1000); $('h1').show(1000);
toggle()	toggle 方法在内部使用 show 和 hide 来改变匹配元素的显示方式，即可见元素将被隐藏，不可见元素将会显示。示例如下。 $('h1').toggle(2000);
slideDown() slideUp(() slideToggle()	类似于 hide 和 show，这些方法隐藏或显示匹配元素。但是，这是通过将元素的 height 从当前尺寸调整为 0，或者从 0 调整为初始尺寸来实现的，所以元素会展开或卷起 slideToggle 方法会展开隐藏的元素，卷起可见的元素，从而可以使用一个动作重复地显示和隐藏元素。示例如下。 $('h1').slideUp(1000); $('h1').slideDown(1000); $('h1').slideToggle(1000);
fadeIn() fadeOut() fadeTo()	这些方法通过修改匹配元素的不透明度来显示或隐藏它们。fadeOut 将不透明度设置为 0，使元素完全透明，然后将 CSS display 属性设置为 none，从而完全隐藏元素。fadeTo 允许指定一个不透明度(0 到 1 之间的一个数字)，以便决定元素的透明程度。fadeIn 使元素再次可见，把不透明度设置为 1。全部 3 个方法都允许定义一个固定速度(慢、中、快)，或者一个定义了动画持续时间(单位为毫秒)的数字。示例如下。 $('h1').fadeOut(1000); $('h1').fadeIn(1000); $('h1').fadeTo(1000, 0.5);
animate()	在内部，animate 用于许多动画方法，如 show 和 hide。但是，也可以在外部使用它，以便更灵活地以动画方式显示匹配元素。使用 animate 方法，可以指定许多动画显示的属性。考虑下面这个示例。 $('h1').animate({ opacity: 0.4, marginLeft: '50px', fontSize: '50px' }, 1500);

(续表)

方　法　名	用　　途
	这段代码接受一个 h1 元素，将其字体大小设置为 50 像素，将其不透明度设置为 0.4 以使元素半透明，并将其左页边距设置为 50 像素，从而在 1.5 秒钟的时间内平滑地进行动画显示。animate 方法的第一个参数是一个对象，它保存一个或者多个想要动画显示的属性，属性之间以逗号分隔。注意，需要使用 JavaScript 的 marginLeft 和 fontSize，而不是 CSS margin-left 和 font-size 属性。只能动画显示接受数值的属性。也就是说，可以使用 margin、fontSize、opacity 等属性，但是不能使用 color 或 fontFamily 这样的属性

下面安的示例将应用两个主要的动画：一个动画在用户提交联系表单时触发，它会逐渐地卷动表单直至其隐藏起来。另一个动画用来显示和隐藏 Message Sent 标记，从而吸引更多注意，直至动画完全消失。

【例 11-5】使用 jQuery 方法实现表格变色显示效果。

(1) 在 WebSite11 网站中，新建一个网页文件 example11-5.aspx，然后添加一个 7 行 4 列的表格，效果如图 11-5 所示。

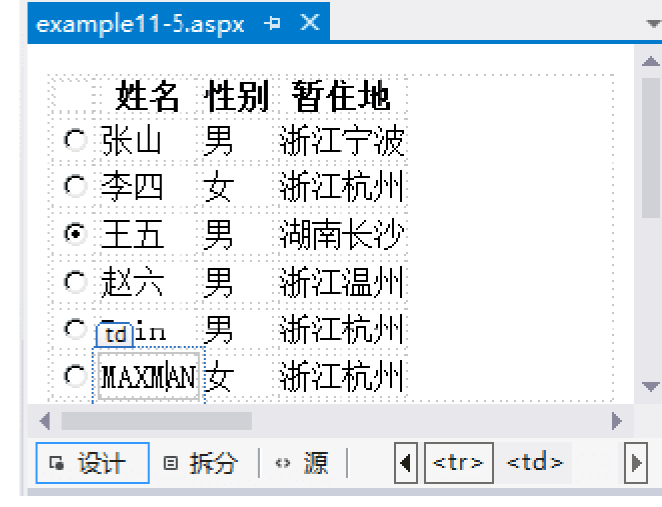


图 11-5　添加表格

生成的代码如下。

```
<table>
    <thead>
        <tr><th> </th><th>姓名</th><th>性别</th><th>暂住地</th></tr>
    </thead>
    <tbody>
        <tr><td><input type="radio" name="choice" value=""/></td>
            <td>张山</td><td>男</td><td>浙江宁波</td></tr>
        <tr><td><input type="radio" name="choice" value="" /></td>
            <td>李四</td><td>女</td><td>浙江杭州</td></tr>
        <tr><td><input type="radio" name="choice" value="" checked='checked' /></td>
            <td>王五</td><td>男</td><td>湖南长沙</td></tr>
        <tr><td><input type="radio" name="choice" value="" /></td>
            <td>赵六</td><td>男</td><td>浙江温州</td></tr>
        <tr><td><input type="radio" name="choice" value="" /></td>
            <td>Rain</td><td>男</td><td>浙江杭州</td></tr>
        <tr><td><input type="radio" name="choice" value="" /></td>
            <td>MAXMAN</td><td>女</td><td>浙江杭州</td></tr>
    </tbody>
</table>
```

(2) 新建一个 css 文件夹，然后新建一个样式文件，命名为 style.css，添加如下代码。

```
table       { border:0;border-collapse:collapse;}
td          { font:normal 12px/17px Arial;padding:2px;width:100px;}
th          { font:bold 12px/17px Arial;text-align:left;padding:4px;border-bottom:1px solid #333;}
.even       { background:#FFF38F;}   /* 偶数行样式*/
```

```
.odd          { background:#FFFFEE;}   /* 奇数行样式*/
.selected     { background:#FF6500;color:#fff;}
```

此代码段主要用于设置表格的显示样式。

(3) 打开页面文件，添加 CSS 文件和 jQuery 文件的引用，代码如下。

```
<!--    引入 CSS 样式和 jQuery -->
<link href="css/style.css" rel="stylesheet" type="text/css" />
<script src="../scripts/jquery-2.1.3.js" type="text/javascript"></script>
```

(4) 在引用之后，添加一个 script 代码段，用于实现表格的变色效果，代码如下。

```
<script type="text/javascript">
 $(function(){
        $("tbody>tr:odd").addClass("odd"); //先排除第一行,然后给奇数行添加样式
        $("tbody>tr:even").addClass("even"); //先排除第一行,然后给偶数行添加样式
        $('tbody>tr').click(function() {
                $(this)
                        .addClass('selected')
                        .siblings().removeClass('selected')
                        .end()
                        .find(':radio').attr('checked',true);
        });
        //  如果单选框默认情况下是选择的，则高色.
         // $('table :radio:checked').parent().parent().addClass('selected');
        //简化:
          $('table :radio:checked').parents("tr").addClass('selected');
            //再简化:

//$('tbody>tr:has(:checked)').addClass('selected')
;
  })
</script>
```

(5) 保存并运行程序，效果如图 11-6 所示。

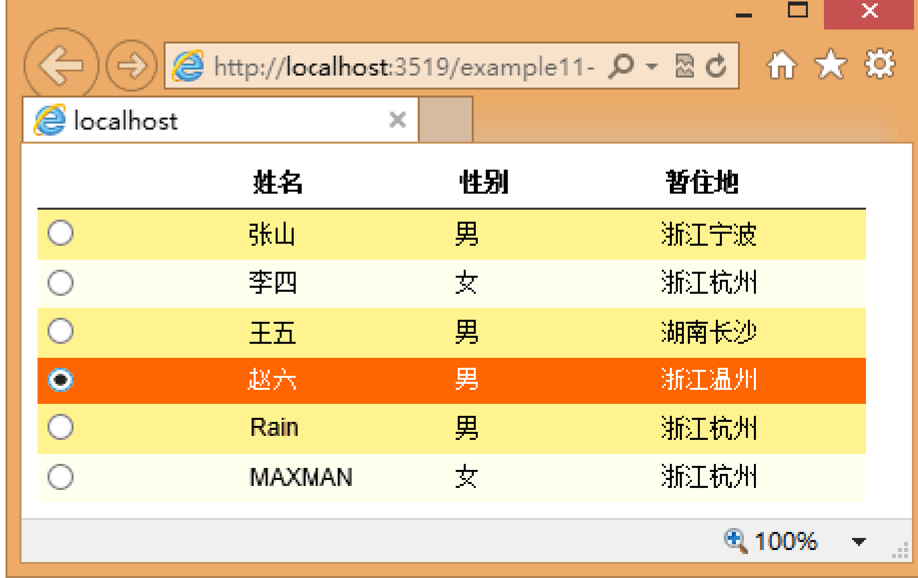


图 11-6　程序运行效果

11.5　jQuery 和有效性验证

jQuery 1.2.6 以上的版本包含有效性验证功能。要使用 jQuery 的验证功能，需要先将 jQuery 库和相关的验证方法库引用应用中，引用格式如下。

```
<script src="../js/jquery.js" type="text/javascript"></script>
<script src="../js/jquery.validate.js" type="text/javascript"></script>
```

jQuery 提供了 17 种默认的校验规则，这些规则如下。

(1) required:true，必输字段。

(2) remote:"check.php"，使用 ajax 方法调用 check.php 验证输入值。

(3) email:true，必须输入正确格式的电子邮件。

(4) url:true，必须输入正确格式的网址。

(5) date:true，必须输入正确格式的日期。

(6) dateISO:true，必须输入正确格式的日期(ISO)，如 2009-06-23，1998/01/22 只验证格式，不验证有效性。

(7) number:true，必须输入合法的数字(负数，小数)。

(8) digits:true，必须输入整数。

(9) creditcard，必须输入合法的信用卡号。

(10) equalTo:"#field"， 输入值必须和#field 相同。

(11) accept，输入拥有合法后缀名的字符串（上传文件的后缀）。

(12) maxlength:5，输入长度最多是 5 的字符串(汉字算一个字符)。

(13) minlength:10，输入长度最小是 10 的字符串(汉字算一个字符)。

(14) rangelength:[5,10]，输入长度必须介于 5 和 10 之间的字符串")(汉字算一个字符)。

(15) range:[5,10]，输入值必须介于 5 和 10 之间。

(16) max:5，输入值不能大于 5。

(17) min:10，输入值不能小于 10。

针对以上验证，jQuery 提供了默认的提示信息。

```
messages: {
    required: "This field is required.",
    remote: "Please fix this field.",
    email: "Please enter a valid email address.",
    url: "Please enter a valid URL.",
    date: "Please enter a valid date.",
    dateISO: "Please enter a valid date (ISO).",
    dateDE: "Bitte geben Sie ein g 眉 ltiges Datum ein.",
    number: "Please enter a valid number.",
    numberDE: "Bitte geben Sie eine Nummer ein.",
    digits: "Please enter only digits",
    creditcard: "Please enter a valid credit card number.",
    equalTo: "Please enter the same value again.",
    accept: "Please enter a value with a valid extension.",
    maxlength: $.validator.format("Please enter no more than {0} characters."),
    minlength: $.validator.format("Please enter at least {0} characters."),
    rangelength: $.validator.format("Please enter a value between {0} and {1} characters long."),
    range: $.validator.format("Please enter a value between {0} and {1}."),
    max: $.validator.format("Please enter a value less than or equal to {0}."),
    min: $.validator.format("Please enter a value greater than or equal to {0}.")
},
```

如果需要修改这些提示信息，可以在 js 代码中加入以下代码。

```
jQuery.extend(jQuery.validator.messages, {
```

```
        required: "必选字段",
        remote: "请修正该字段",
        email: "请输入正确格式的电子邮件",
        url: "请输入合法的网址",
        date: "请输入合法的日期",
        dateISO: "请输入合法的日期 (ISO).",
        number: "请输入合法的数字",
        digits: "只能输入整数",
        creditcard: "请输入合法的信用卡号",
        equalTo: "请再次输入相同的值",
        accept: "请输入拥有合法后缀名的字符串",
        maxlength: jQuery.validator.format("请输入一个长度最多是 {0} 的字符串"),
        minlength: jQuery.validator.format("请输入一个长度最少是 {0} 的字符串"),
        rangelength: jQuery.validator.format("请输入一个长度介于 {0} 和 {1} 之间的字符串"),
        range: jQuery.validator.format("请输入一个介于 {0} 和 {1} 之间的值"),
        max: jQuery.validator.format("请输入一个最大为 {0} 的值"),
        min: jQuery.validator.format("请输入一个最小为 {0} 的值")
    });
```

建议将此文件放入单独的 js 文件中，如放入 messages_cn.js 中，然后在页面中引入，引入代码如下。

```
    <mce:script src="../js/messages_cn.js" mce_src="js/messages_cn.js"
type="text/javascript"></mce:script>
```

在使用 jQuery 验证功能时，有两种使用方式。

1．将校验规则写到控件中

将校验规则写到控件中，示例代码如下。

```
    <mce:script src="../js/jquery.js" mce_src="js/jquery.js" type="text/javascript"></mce:script>
    <mce:script src="../js/jquery.validate.js" mce_src="js/jquery.validate.js"
type="text/javascript"></mce:script>
    <mce:script src="./js/jquery.metadata.js" mce_src="js/jquery.metadata.js"
type="text/javascript"></mce:script>
    $().ready(function() {
     $("#signupForm").validate();
    });

    <form id="signupForm" method="get" action="">
        <p>
            <label for="firstname">Firstname</label>
            <input id="firstname" name="firstname" class="required" />
        </p>
     <p>
      <label for="email">E-Mail</label>
      <input id="email" name="email" class="required email" />
     </p>
     <p>
      <label for="password">Password</label>
      <input id="password" name="password" type="password" class="{required:true,minlength:5}" />
```

```
        </p>
        <p>
          <label for="confirm_password">确认密码</label>
          <input id="confirm_password" name="confirm_password" type="password"
class="{required:true,minlength:5,equalTo:'#password'}" />
        </p>
            <p>
                <input class="submit" type="submit" value="Submit"/>
            </p>
    </form>
```

使用 class="{}"的方式，必须引入包 jquery.metadata.js。可以使用如下的方法，修改提示内容。

```
    class="{required:true,minlength:5,messages:{required:'请输入内容'}}"
```

在使用 equalTo 关键字时，后面的内容必须加上引号，如下代码。

```
    class="{required:true,minlength:5,equalTo:'#password'}"
```

另外一个方式，使用关键字 meta，代码如下。

```
    meta: "validate"
    <input id="password" name="password" type="password"
class="{validate:{required:true,minlength:5}}" />
```

还有一种方式，代码如下。

```
    $.metadata.setType("attr", "validate");
```

这样可以使用 validate="{required:true}"的方式，或 class="required"，但 class="{required:true,minlength:5}"将不起作用。

2.将校验规则写到代码中

第二种使用 jQuery 验证功能的方法是将校验规则写到代码中，示例代码如下。

```
    $().ready(function() {
     $("#signupForm").validate({
            rules: {
       firstname: "required",
       email: {
        required: true,
        email: true
       },
       password: {
        required: true,
        minlength: 5
       },
       confirm_password: {
        required: true,
        minlength: 5,
```

```
            equalTo: "#password"
          }
        },
                messages: {
          firstname: "请输入姓名",
          email: {
            required: "请输入 Email 地址",
            email: "请输入正确的 email 地址"
          },
          password: {
            required: "请输入密码",
            minlength: jQuery.format("密码不能小于{0}个字符")
          },
          confirm_password: {
            required: "请输入确认密码",
            minlength: "确认密码不能小于 5 个字符",
            equalTo: "两次输入密码不一致"
          }
        }
          });
    });
    //messages 处，如果某个控件没有 message，将调用默认的信息

    <form id="signupForm" method="get" action="">
        <p>
            <label for="firstname">Firstname</label>
            <input id="firstname" name="firstname" />
        </p>
      <p>
        <label for="email">E-Mail</label>
        <input id="email" name="email" />
      </p>
      <p>
        <label for="password">Password</label>
        <input id="password" name="password" type="password" />
      </p>
      <p>
        <label for="confirm_password">确认密码</label>
        <input id="confirm_password" name="confirm_password" type="password" />
      </p>
        <p>
            <input class="submit" type="submit" value="Submit"/>
        </p>
    </form>
```

通过这种方式使用 jQuery 验证功能，需注意以下几点。

- required:true 必须有值。
- required:"#aa:checked"表达式的值为真，则需要验证。
- required:function(){}返回为真，表示需要验证。
- 后边两种常用于，表单中需要同时填或不填的元素。

11.6　关于 jQuery 的实用提示

为了最大程度地利用 jQuery，应该遵循下面的提示。

- 不断试验。一开始时，jQuery 是一个有点奇怪的技术，不太容易掌握，这主要是因为它所使用的大量大括号和小括号。但是，通过不断尝试，开发者很快就可以成为 jQuery 高手。
- 访问 jQuery.com 网站。除了提供优质文档和许多显示 jQuery 功能的示例之外，还可以在站点上找到大量关于使用 jQuery 的文章和链接，包括提供视频内容的站点链接。
- 投入一些时间浏览 jQuery UI 和 Juice UI 网站。开发者可能不会马上用到插件，或者可能还没有准备进行下一步，但是在建立下一个有丰富客户端界面的 Web 站点时，应该知道它们可以提供的功能。

11.7　本章小结

本章介绍了 jQuery，这是一种非常流行的开源客户端 JavaScript 框架，可以用来与文档对象模型进行交互。首先介绍了 jQuery 的下载和将其添加到 Web 站点中的方法；然后通过示例介绍了 jQuery 在 ASP.NET 中的使用方法；之后介绍了在页面中定位相关元素的 jQuery 选择器和筛选器。

接下来讲解了可以对匹配集应用效果和动画的多种 jQuery 方法，了解如何使用 css 这样的方法来操作 CSS 设置，如何使用 parent 和 prev 方法导航匹配集中的项，以及如何通过事件在发生某些动作时触发代码。

通过本章的学习，读者能够学习到如何使用 jQuery 中的众多动画方法，以使页面的外观更具吸引力，并且交互性更好，以及如何在 ASP.NET 验证框架中使用 jQuery 的功能。

11.8　思考和练习

1. 利用 Timer 控件和页面局部更新技术，实现广告显示一段时间后自动关闭的功能。
2. 假设在服务器端每隔 1 秒处理一次业务，10 秒钟处理完毕。设计一个页面，利用局部更新技术，用进度条自动显示每次处理的进度。
3. 编写一个了解学生业余活动的调查页面，页面中包含对输入信息的验证功能。程序运行效果如图 11-7 所示。



图 11-7　练习 3 页面设计效果

第12章　Web服务

在井喷式发展的移动互联时代，在应用程序中更好地整合和充分、快捷、有效地利用Internet上的海量计算能力和海量存储信息，实现在因特网上建立广泛交互服务的“五个任何”(任何人，任何地方，任何时间，任何设备，任何手段)互联梦想，使Internet成为一个透明的超级计算机是开发者的梦想。这个迷人的梦想的实现者就是以XML(eXtensible Markup Language)或JSON(JavaScript Object Notation)为基础的数据交换格式、支持Ajax (Asynchronous JavaScript and XML)的Web Services(Web服务)。

支持Ajax的Web服务是目前最新的技术，它可以轻松整合不同的应用程序以及异构系统之间的数据共享。本章首先介绍Web服务的基本概念和特点，然后通过具体实例说明如何创建和使用支持Ajax的Web服务。

本章的学习目标：

- 了解什么时候需要使用Web服务。
- 掌握如何创建和调用支持Ajax的Web服务。

12.1　支持AJAX的Web Services简介

ASP.NET AJAX提供了完整的架构，允许从客户端JavaScript调用ASP.NET Web Services。设计者可以轻松地用AJAX把服务器端的数据和功能集成到用户响应的Web页面中，而所需要做的就是仅仅用[ScriptService]属性来标识Web Service。ASP.NET AJAX框架会为Web Service自动生成JavaScript代理，然后通过使用代理来调用Web methods。

12.1.1　什么是Web Services

简单地讲，Web Services(Web服务)是一个基于因特网的可通过Web被远程调用的应用程序模块(API)。例如，网站里想提供天气预报的服务，不用自己实现天气预报的功能，只用调用其他公司提供的免费或付费Web Services即可。

- 服务就是一个软件，它和客户端应用程序没有很紧密地耦合或关联。服务是可以被动态地发现及组合成其他软件的软件实体。
- Web服务是一种基于XML、JSON、SOAP、HTTP、UDDI、WSDL等一系列标准实现的分布式计算技术和软件组件。
- Web服务提供了一个松散耦合和跨平台的分布式计算环境，它是一个与操作系统无关、程序设计语言无关、机器类型无关、运行环境无关的平台，实现网络上应

用的共享，并可用于复杂的系统集成。

- 国际标准化组织 W3C：Web Service 是一个通过 URL 识别的软件应用程序，其界面及绑定能用 XML 文档来定义、描述和发现，使用基于 Internet 协议上的消息传递方式与其他应用程序进行直接交互。
- Microsoft：Web Service 是为其他应用提供数据和服务的应用逻辑单元，应用程序通过标准的 Web 协议和数据格式获得 Web Service，如 HTTP、XML/JSON 和 SOAP 等，每个 Web Service 的实现是完全独立的。Web Service 具有基于组件的开发和 Web 开发两者的优点，是 Microsoft 的.Net 程序设计模式的核心。

Web Service 是一种革命性的分布式计算技术，它使用基于 XML/JSON 的消息处理作为基本的数据通信方式，消除使用不同组件模型、操作系统和编程语言的系统之间存在的差异，使异类系统能够作为计算网络的一部分协同运行。开发人员可以使用像过去创建分布式应用程序时使用组件的方式，创建由各种来源的 Web 服务组合在一起的应用程序。由于 Web 服务是建立在一些通用协议的基础上，如 HTTP、SOAP、XML、JSON、WSDL、UDDI 等，这些协议在涉及到操作系统、对象模型和编程语言的选择时，没有任何倾向，因此 Web 服务将会有很强的生命力。

SOAP，即 Simple Object Access Protocol，简单对象访问协议。SOAP 是消息传递的协议，它规定了 Web Services 之间是怎样传递信息的。简单地说，SOAP 规定了以下几点规则。

(1) 传递信息的格式为 XML。这就使 Web Services 能够在任何平台上，用任何语言进行实现。

(2) 远程对象方法调用的格式。规定了怎样表示被调用对象以及调用的方法名称和参数类型等。

(3) 参数类型和 XML 格式之间的映射。这是因为，被调用的方法有时候需要传递一个复杂的参数，如一个 Person 对象。怎样用 XML 来表示一个对象参数，也是 SOAP 所定义的范围。

WSDL，即 Web Services Description Language，Web 服务描述语言。WSDL 是 Web Services 的定义语言。当实现了某种服务的时候(如股票查询服务)，为了让别的程序调用，必须告诉大家服务接口，如服务名称、服务所在的机器名称、监听端口号、传递参数的类型、个数和顺序、返回结果的类型等。这样别的应用程序才能调用该服务。WSDL 协议就是规定了有关 Web Services 描述的标准。

UDDI，即 Universal Description Discovery and Integration，通用描述发现和集成协议。简单说，UDDI 用于集中存放和查找 WSDL 描述文件，起着目录服务器的作用。

扩展标记语言 XML，是 Internet 上数据表示和数据交换的新标准。是 SGML(Standard for General Markup Language，即：通用标记语言标准)的一个简化子集。XML 关注信息本身，是 Web 上表示结构化信息的一种标准文本格式。与传统的注重页面信息显示的 HTML(Hypertext Markup Language，即：超文本链接标示语言)相比，关注于内容的 XML 具有以下诸多优点：良好的可扩展性，语言简单有效，可自行定义标记；内容与形式的分离，主要刻画数据内容，不考虑显示效果；有严格的语法要求，便于分析统一和与数据库

信息转换；便于传输，为纯文本形式，可通过 Http 协议直接传输，可跨越防火墙等。XML 的出现和发展对于 Internet 和 Intranet 产生了巨大的影响。

1. Web 服务的目标

(1) 让地理上分布在不同区域的计算机和设备一起工作，为用户提供各种各样的服务。

(2) 用户可以控制要获取信息的内容、时间、方式，而不必像目前这样，在无数个信息孤岛中浏览，去寻找自己需要的信息。

2. Web 服务的影响

(1) Web 服务支持在 Web 站点上放置可编程的元素，用户可以抓取已有的元素，构成自己的新服务。

(2) 能进行基于 Web 的分布式计算和处理，能很好地兼容现有的 Web 技术。

(3) Web 服务使得 Internet 成为一个可以无限扩展、拥有无限潜力的分布式计算平台。

(4) 任何设备可以随时随地访问 Internet 上的 Web 服务。

(5) 软件模块充分复用、计算机资源充分共享、信息无缝共享和交流。

3. Web 服务的好处

(1) Web 服务改变企业对 IT 系统的应用方式和依赖程度。在 Web 服务的帮助下，企业可以不再耗费巨资建设 IT 系统，转而使用 Internet 上提供的各类 Web 服务来实现。

(2) 利用 Web 服务，公司和个人将能够迅速且廉价地向整个国际互联网络提供他们的服务，进而建立全球范围的联系，在广泛的范围内寻找可能的合作伙伴。

4. Web 服务的主要特征

(1) 互操作性：一个 Web 服务可与其他 Web 服务交互，协同工作；可以使用任何语言开发 Web 服务或使用他人提供的 Web 服务；开发环境可以异构。

(2) 普遍性：Web 服务使用 HTTP 和 XML 进行通信，支持这些技术的设备都可以拥有和访问 Web 服务。

(3) 松散耦合：Web 服务的实现对使用者透明，当服务的实现发生变动时不影响用户使用。

(4) 高度可集成能力：Web 服务和采用了简单的、易理解的标准 Web 协议作为组件界面描述和协同描述规范，屏蔽了平台的异构性，CORBA、DCOM、EJB 等都可通过它进行互操作。

5. Web 服务架构

SOA(Service Oriented Architecture)架构如图 12-1 所示。其中包括了 3 个参与者和 3 个基本操作。

- 3 个参与者：服务提供者、服务请求者、服务中介者。
- 3 个基本操作：发布、发现、绑定。

XML 或 JSON 是数据的格式，SOAP 是调用 Web Services 的协议，WSDL 是描述 Web Services 的格式，而 UDDI 是 Web Services 发布、查找和利用的组合。

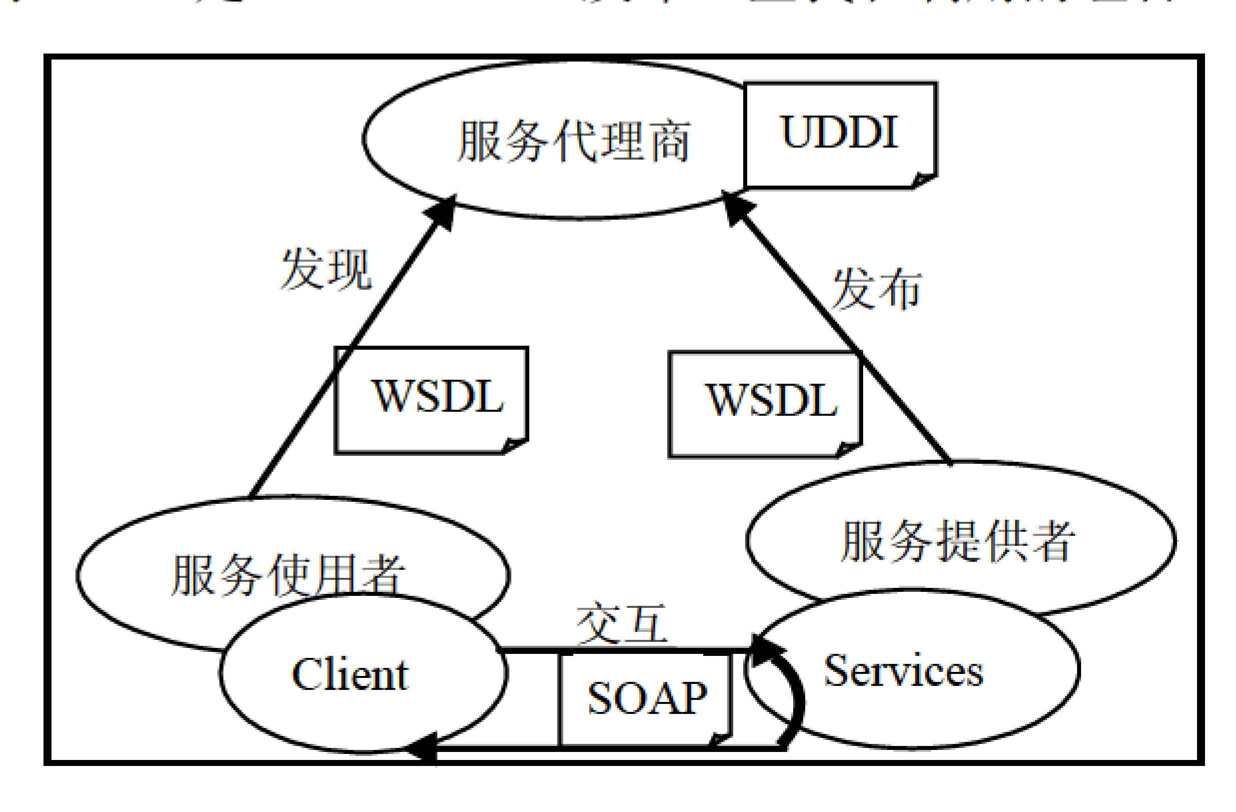


图 12-1　SOA Web 服务架构

6. Web 服务有两种类型的应用

(1) 可重复使用的应用程序组件

有一些功能是不同的应用程序常常会用到的。那么为什么要周而复始地开发它们呢？

Web services 可以把应用程序组件作为服务来提供，如汇率转换、天气预报或者甚至是语言翻译等。比较理想的情况是，每种应用程序组件只有一个最优秀的版本，这样任何人都可以在其应用程序中使用它。

(2) 连接现有的软件

通过为不同的应用程序提供一种链接其数据的途径，Web Services 有助于解决协同工作的问题。通过使用 WebServices，用户可以在不同的应用程序与平台之间来交换数据。

12.1.2　JSON 和支持 AJAX 的 Web 服务

JSON(JavaScript Object Notation)是一种轻量级的数据交换格式，易于阅读和编写，易于机器解析和生成。它基于 JavaScript Programming Language，Standard ECMA-262 3rd Edition - December 1999 的一个子集。JSON 采用完全独立于语言的文本格式，使用了类似于 C 语言家族的习惯(包括 C，C++，C#，Java，JavaScript，Perl，Python 等)。这些特性使 JSON 成为理想的数据交换语言。

JSON 建构于两种结构：

(1) “名称/值”对的集合。不同的语言中，它被理解为对象(object)、记录(record)、结构(struct)、字典(dictionary)、哈希表(hash table)、有键列表(keyed list)或者关联数组(associative array)。

(2) 值的有序列表。在大部分语言中，它被理解为数组(array)。

这些都是常见的数据结构。事实上大部分现代计算机语言都以某种形式支持它们。这使得一种数据格式在同样基于这些结构的编程语言之间交换成为可能。

JSON 以一种特定的字符串形式来表示 JavaScript 对象。如果将具有这样一种形式的字

符串赋给任意一个 JavaScript 变量，那么该变量会变成一个对象引用，而这个对象就是字符串所构建出来的，好像有点拗口，还是用实例来说明。

这里假设需要创建一个 User 对象，并具有用户 ID、用户名、用户 Email 这 3 个属性，可以使用以下 JSON 形式来表示 User 对象。

```
{"UserID":111, "Name":"Landy", "Email":Landy@gmail.com"};
```

接下来，如果把这一字符串赋予一个 JavaScript 变量，那么就可以直接使用对象的任一属性了。完整代码如下。

```
<script>
    var User = {"UserID":111, "Name":"Landy", "Email":":Landy@gmail.com"};
    alert(User.Name);
</script>
```

借助 ASP.NET AJAX Extension，微软选择 JSON 在服务器和 Ajax 客户端实现数据交换，从而创建支持 AJAX 的 Web 服务。在客户端和服务器端都实现了(数据的)串行化器和并行化器以使数据按 JSON 的格式交换。网页中的客户端脚本与服务器通过 Web 服务通信层进行通信来访问 Web 服务，该通信层使用 AJAX 技术进行 Web 服务调用，数据在客户端与服务器之间通常采用 JSON 格式进行异步交换。

通常，默认将JavaScript对象序列化为JSON格式，使用JavaScript eval函数可进行反序列化。但Web服务和ASP.NET网页中的单个方法可以返回其他格式(如XML)。可以通过ScriptMethod属性来指定方法的序列化格式。对于某个ASMX服务，可以设置ScriptMethod属性形如[ScriptMethod(ResponseFormat = ResponseFormat.Xml)]，以使某个Web服务方法返回XML数据。

12.2 创建和使用支持 AJAX 的 Web 服务

使用 VS 可以创建能够从客户端脚本访问的 ASP.NET 自定义 Web 服务(.asmx 文件)。在【例 12-1】中，将创建一个支持 AJAX 的 Web 服务，并使用客户端脚本调用其方法。

【例 12-1】在 VS 中创建支持 AJAX 的 Web 服务。注意，要完成本示例，需要在本地计算机上安装 Microsoft Internet 信息服务(IIS)。

(1) 创建支持 AJAX 的 Web 服务的网站。需要注意的是，在本实例中必须使用 IIS 网站。

① 打开 VS。

② 选择【文件】|【新建】|【网站】命令，打开【新建网站】对话框，VS 将只筛选列出可以新建的网站项目。

③ 在对话框左侧的【模板】下选择 Visual C#，然后在右侧选择【ASP.NET 空网站】，单击【浏览】按钮打开【选择位置】对话框，单击对话框左侧的【本地 IIS】，单击【默认

网站】，在右上角单击【创建新 Web 应用程序】图标，如图 12-2 所示。VS 创建了一个新的 IIS Web 应用程序。

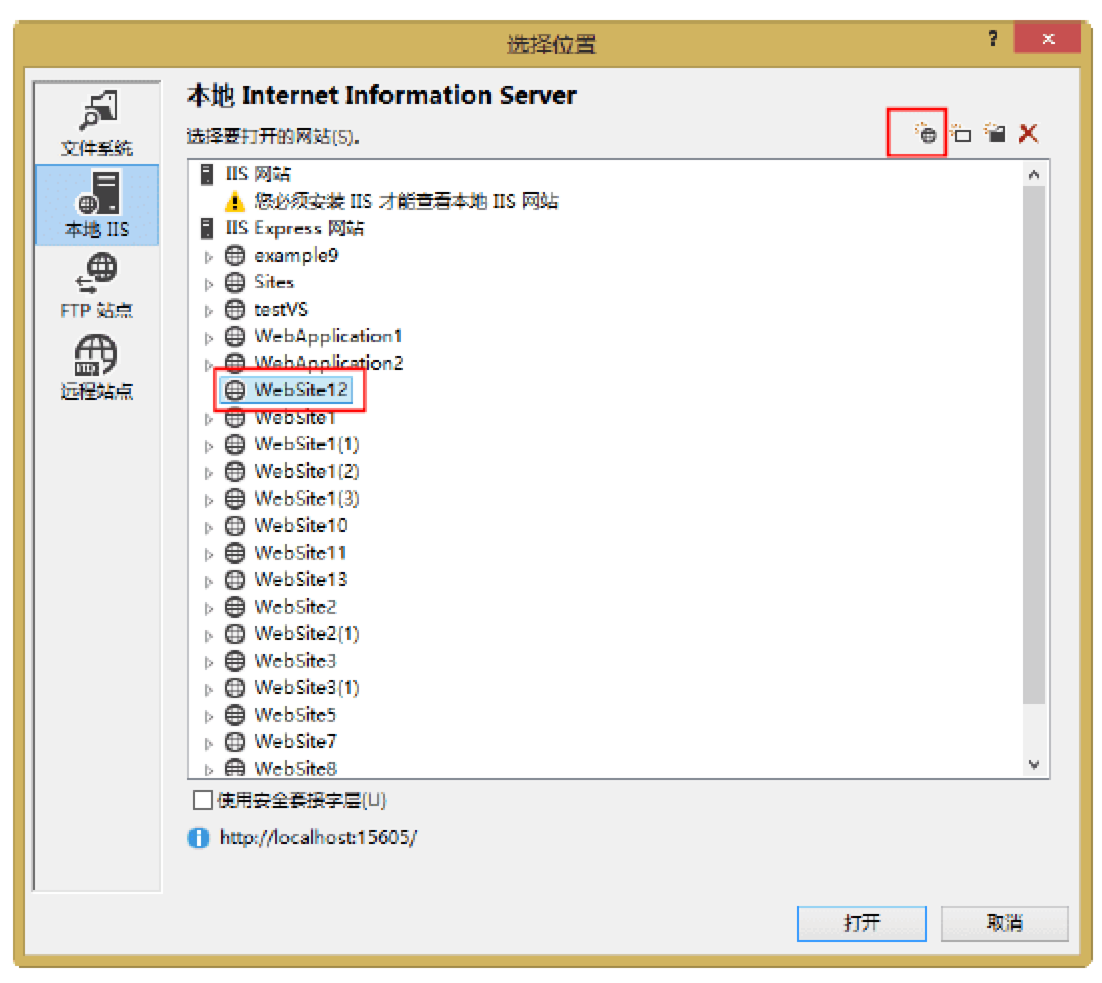


图 12-2　【选择位置】对话框

指定名称 AJAXsupportedWebService，不选择【使用安全套接字层(U)】。

单击【打开】按钮，出现【新建网站】对话框，新网站的名称位于【位置】列表中，包括协议 http://和位置 localhost，如 http://localhost:15605/ (这指示处理本地 IIS 网站)。在【语言】列表中，选择编程语言 Visual C#，单击【确定】按钮。

(2) 编写支持 AJAX 的自定义 Web 服务。

① 在【解决方案资源管理器】中，右击网站名称，选择【添加】|【添加新项】命令。

② 在 Visual Studio【已安装】的 Visual C#模板下，单击【Web 服务】，然后在【名称】文本框中输入 GreetingsWebService.asmx，如图 12-3 所示。

③ 确保【将代码放在单独的文件中】复选框已选定，然后单击【添加】按钮。

VS 创建一个新的 Web 服务，它由两个文件组成：GreetingsWebService.asmx 文件是用来调用 Web 服务方法的文件，它指向 Web 服务代码；代码本身位于 App_Code 文件夹中的类文件(GreetingsWebService.cs)中，代码文件包含 Web 服务的模板。

④ 打开文件 GreetingsWebService.asmx，修改为如下代码。

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/GreetingsWebService.cs"
        Class=" Samples.Aspnet.GreetingsWebService" %>
```

其中，@WebService 指令表示在该文件中声明一项 Web 服务，Language 属性指明使用的语言是 Visual C#，CodeBehind 属性指明实现该服务的代码文件是~/App_Code/GreetingsWebService.cs，Class 属性指明实现该 Web 服务的类的类名是 GreetingsWebService。

⑤ 打开文件 GreetingsWebService.cs，可看到如下代码。将 1~20 行代码用 namespace Samples.Aspnet 名称空间命令括起来。

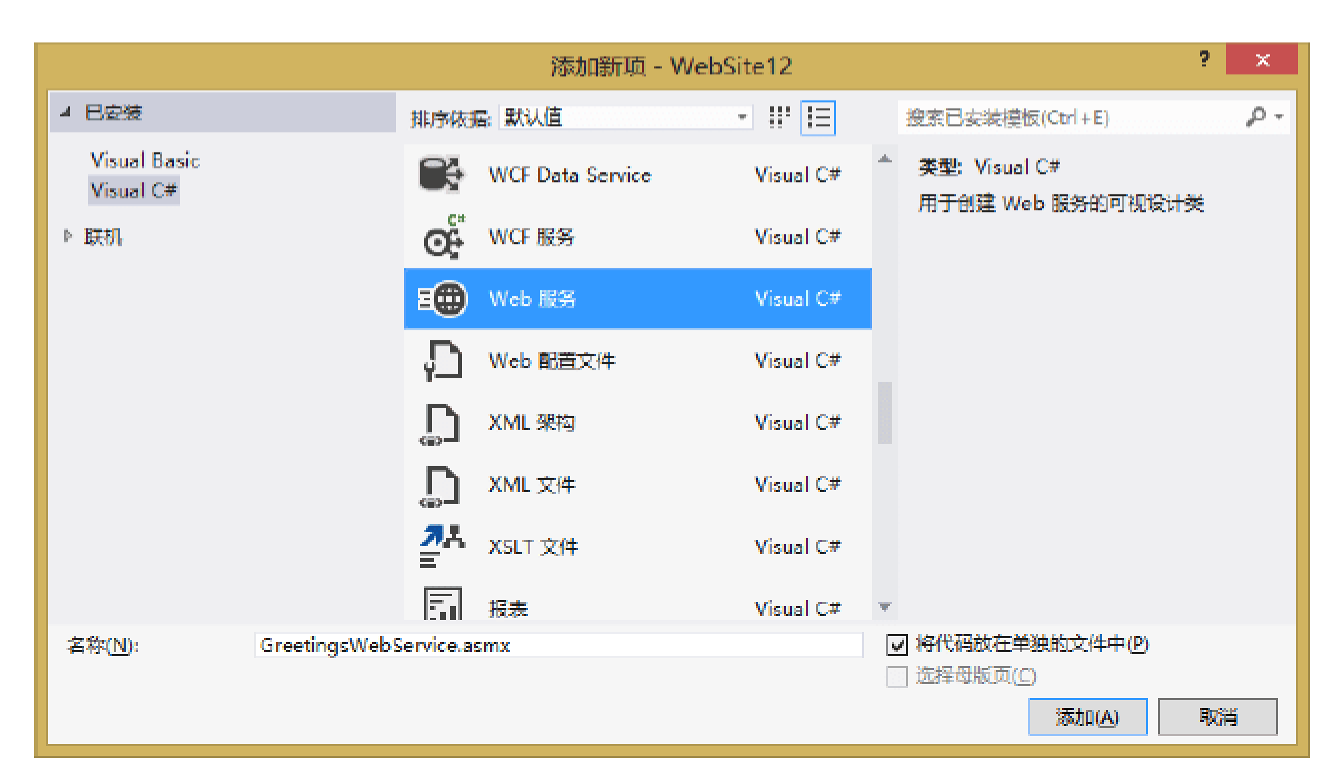


图 12-3　【添加新项】对话框

```
namespace Samples.Aspnet
{
1       /// <summary>
2       ///GreetingsWebService 的摘要说明
3       /// </summary>
4       [WebService(Namespace = "http://tempuri.org/")]
5       [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1) ]
6       //若要允许使用 ASP.NET AJAX 从脚本中调用此 Web 服务，请取消对下行的注释。
7       [System.Web.Script.Services.ScriptService]
8       public class GreetingsWebService : System.Web.Services.WebService
9       {
10          public GreetingsWebService ()//类的构造函数
11          {
12              //如果使用设计的组件，请取消注释以下行
13              //InitializeComponent();
14          }
15          [WebMethod]
16          public string HelloWorld()
17          {
18              return "Hello World";
19          }
20      }
21  }
```

代码的第 8~10 行定义了一个名为 GreetingsWebService 的类，从 WebService 类派生。使用 ASP.NET 创建的 Web 服务的类可以选择从 WebService 类派生，以获得对通用 ASP.NET 对象的访问权限，如 Application、Session、User 和 Context。

代码的第 4 行是应用 WebService 属性设置 Web 服务所属的 XML 命名空间。

将可选的 WebService 属性应用于实现 Web 服务的类，以便为该 Web 服务设置默认的 XML 命名空间(该命名空间原来是 http://tempuri.org)，同时还用一个字符串来描述该 Web 服务。

在使该 XML Web Services 可以公开使用之前，强烈建议修改此默认命名空间 http://tempuri.org。这一点很重要，因为必须将该 XML Web Services 与其他可能不慎将该命名空间用作默认命名空间(<http://tempuri.org/>)的 XMLWeb Services 相区分。

代码的第 15~19 行定义 Web 服务方法。实现 Web 服务的类的方法不会自动具有通过 Web 进行通信的能力，但通过使用 ASP.NET 创建的 Web 服务，很容易添加该功能。若要添加此功能，请将 WebMethod 属性应用于公共方法。可通过 Web 进行通信的 Web 服务的方法称为 Web 服务方法。

请注意，函数名 HelloWorld 前面放置了[WebMethod]属性，作为函数声明的一部分。此外，类 HelloWorld 由[ScriptService]属性限定。利用这些属性，可以从支持 AJAX 的 ASP.NET 网页中的脚本调用 Web 服务。

(3) 测试 Web 服务。

现在就可以在浏览器中测试 Web 服务，此测试不使用脚本来调用 Web 服务方法。测试的目的仅仅是验证 Web 服务是否正常运行。测试步骤如下。

① 打开浏览器，输入以下 URL。

```
http://localhost:15605/ GreetingsWebService.asmx.asmx
```

来调用 Web 服务，在浏览器中会显示一个页，显示由 Web 服务公开的方法。

或者，确保当前文件是 GreetingsWebService.asmx，按 Ctrl+F5 组合键，可以在浏览器中看到该程序运行的界面，如图 12-4 所示。

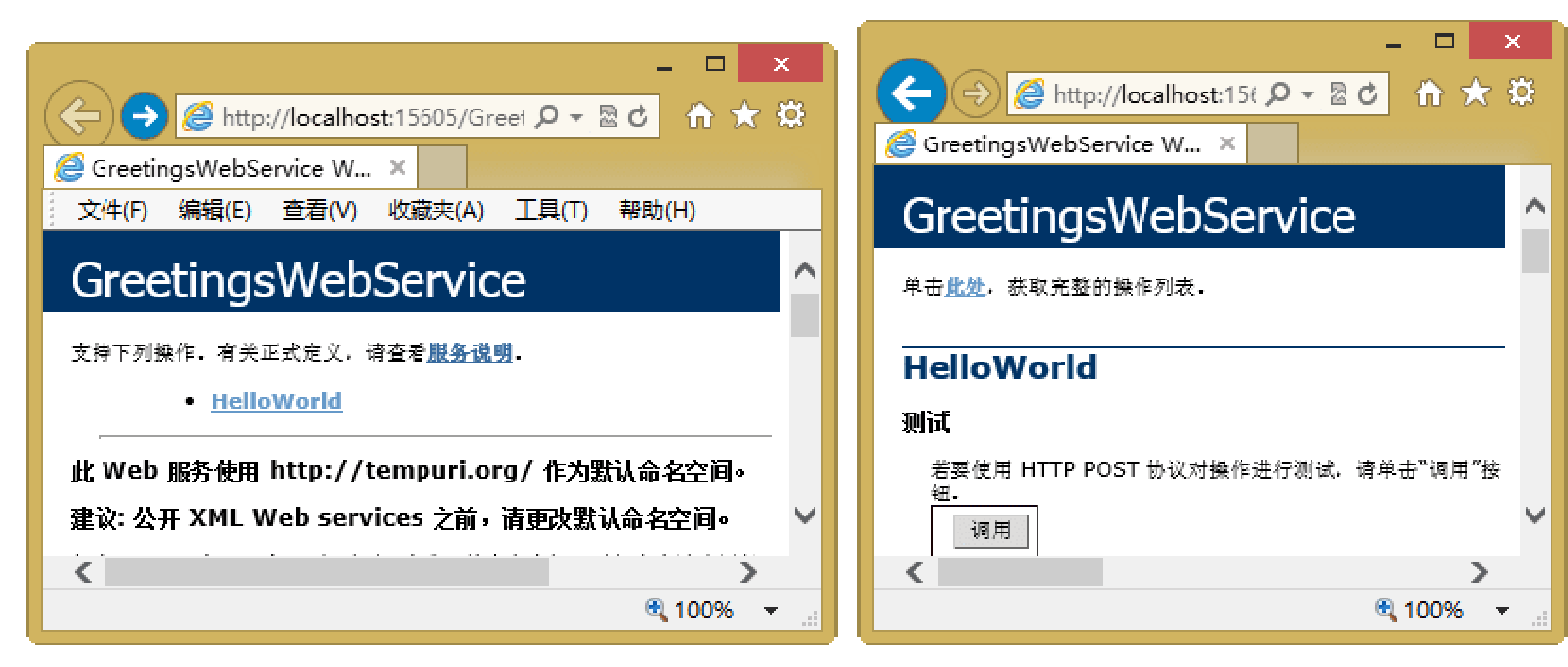


图 12-4　GreetingsWebService.asmx 运行示意图和调用 Web 服务 Hello 的界面

② 单击 HelloWorld。出现一个带有【调用】(Invoke)按钮的页。

③ 单击【调用】按钮调用方法，可看到如图 12-5 所示的 XML 格式的结果。

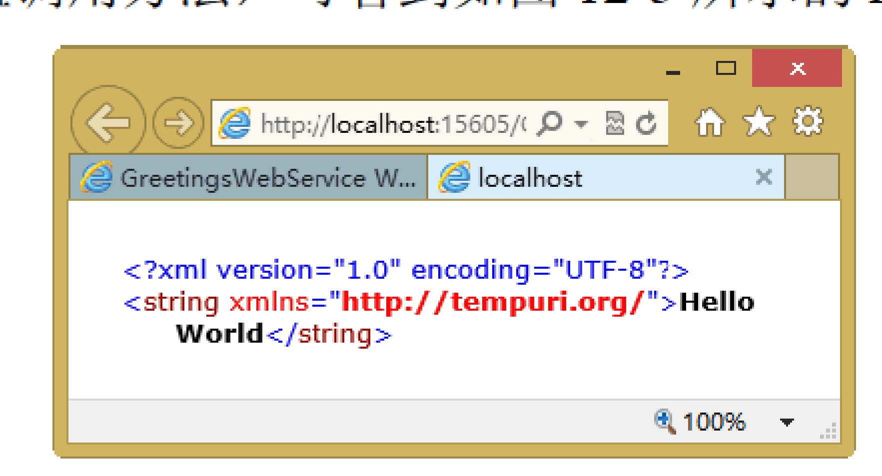


图 12-5　HelloWorld 服务的返回结果

④ 单击图 12-4 中的【服务说明】链接，将看到用 WSDL 语言描述该 Web 服务的 XML 文档。

⑤ 关闭浏览器。保存文件并将其关闭，完成创建支持 AJAX 的 Web 服务。

(4) 打开文件 GreetingsWebService.cs，添加 Web 服务方法，代码如下。

```
[WebMethod(Description = "无参数的 Web 服务，显示“Hello world！和服务器的当前时间”")]
public string Greetings()
{
    string serverTime = String.Format("Current date and time: {0}.", DateTime.Now);
    string greetings = "Hello World! <br/>" + serverTime;
    return greetings;
}
[WebMethod(Description = "有参数的 Web 服务，显示“Hello world 您输入的用户名！"
    +"和服务器的当前时间”")]
public string GreetingsParamters(string vistorName)
{
    string inputString = Server.HtmlEncode(vistorName);
    if (!String.IsNullOrEmpty(inputString)) //if (vistorName.Trim().Length != 0)//also ok.
    {
        string serverTime =String.Format("Current date and time is {0}.", DateTime.Now);
        string greetings = "Hello, " + inputString + "!<br/>" + serverTime;
        return greetings;
    }
    else return "my friend,please enter your name!";
}
```

(5) 从客户端脚本调用 Web 服务。

① 在【解决方案资源管理器】中，右击网站名称，然后选择【添加】|【添加新项】命令，打开【添加新项】对话框。

② 在已安装的模板下，单击【JavaScript 文件】，然后在【名称】文本框中输入 callGreetingsClientJScript.js，单击【确定】按钮。

③ 向脚本文件中添加以下代码。

```
var GreetingsServiceProxy;// Initializes global and proxy default variables.
function pageLoad()
{
    // Instantiate the service proxy.
    GreetingsServiceProxy = new   Samples.Aspnet.GreetingsWebService();
    // Set the default call back functions.
    GreetingsServiceProxy.set_defaultSucceededCallback(SucceededCallback);
    GreetingsServiceProxy.set_defaultFailedCallback(FailedCallback);
    GreetingsServiceProxy.set_timeout(2000);
    GreetingsServiceProxy.set_defaultUserContext("my context");
}
// Processes the button click and calls the service Greetings method.处理按钮事件，调用 Web 服务
function OnClickGreetings()
{
    var greetings =   GreetingsServiceProxy.Greetings();
}
// Processes the button click and calls the service Greetings method with parameters.
function OnClickGreetingsParameters(s) //处理按钮事件，调用带参数的 Web 服务
{
  var greetings =GreetingsServiceProxy.GreetingsParamters(s,SucceededCallback,FailedCallback);
}
function SucceededCallback(result) // Callback function that processes the service return value.
```

```
{
    var RsltElem = document.getElementById("Results");
    RsltElem.innerHTML = result;
}
// Callback function invoked when a call to the service methods fails.
function FailedCallback(error, userContext, methodName)
{
    if(error !== null)
    {
        var RsltElem = document.getElementById("Results");
        RsltElem.innerHTML = "An error occurred: " + error.get_message();
    }
}
if (typeof(Sys) !== "undefined") Sys.Application.notifyScriptLoaded();
```

在调用某个 Web 服务时，通常定义两个回调：一个用于成功的情况；另一个用于失败的情况。Web 服务调用是异步执行的，必须通过回调来了解调用的实际完成情况。这里的方法 SucceededCallback 和 FailedCallback 分别实现成功回调和失败回调。

新建并打开 callGreetingsService.aspx 页面，将使用此客户端脚本来调用 GreetingsWebService.cs中定义的Greetings()和GreetingsParamters()这两个Web服务方法。

④ 切换到 callGreetingsService.aspx 的【设计】视图，从工具箱的【AJAX 扩展】组中向页面中拖放一个 ScriptManager 控件，使用默认的控件名称，然后设置 ScriptManager 控件的 Services 属性。需要从客户端代码中用 ScriptManager 注册需要与之交互的服务。单击 Services 属性右侧的【浏览】按钮，打开【ServiceReference 集合编辑器】对话框，单击【添加】按钮，在右侧设置 path 为 GreetingsWebService.asmx，即完成向 ScriptManager 控件中添加服务引用，如图 12-6 所示。同理，为 Scripts 属性添加对脚本文件的引用路径 path 为 callGreetingsClientJScript.js。

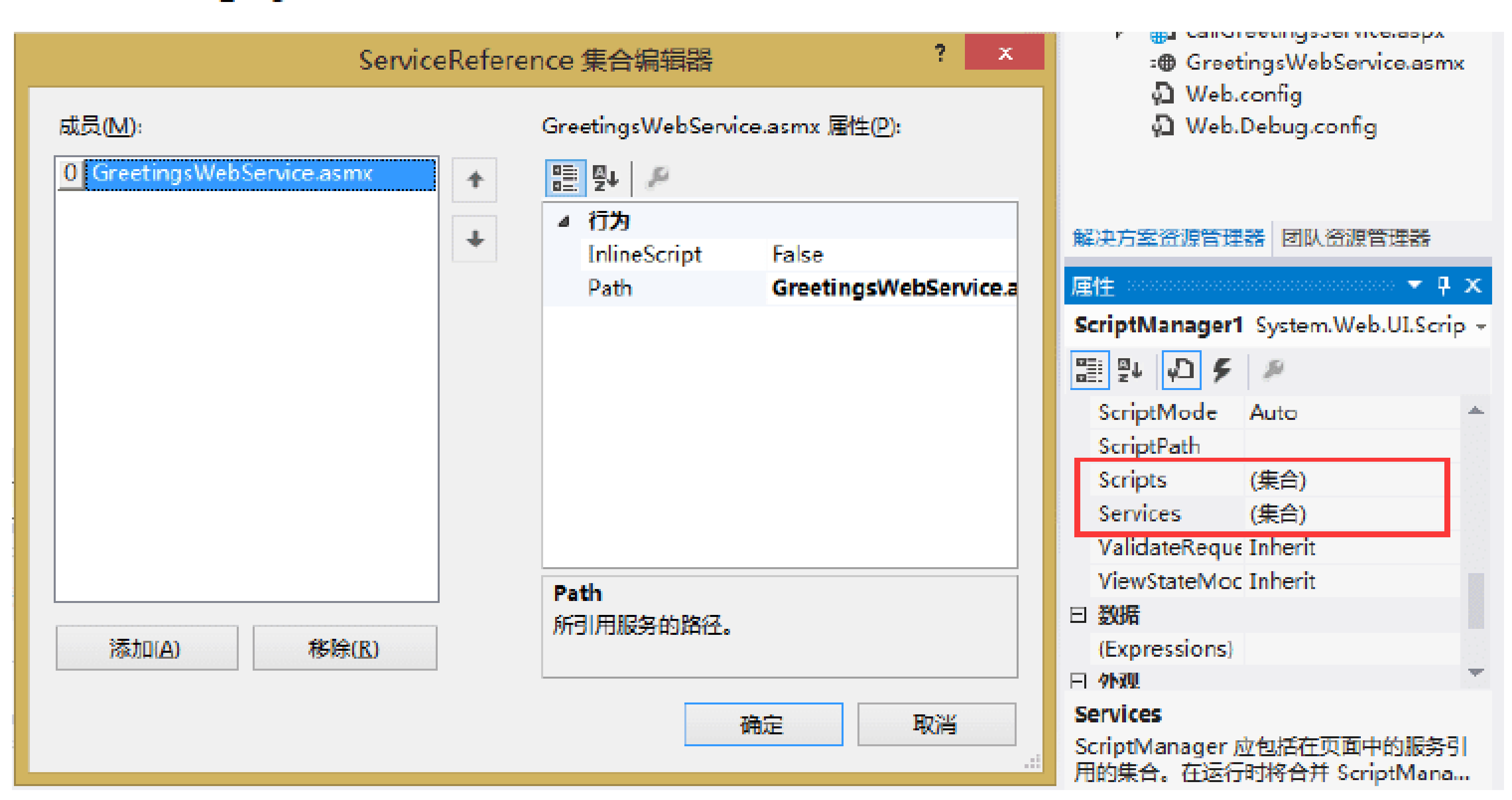


图 12-6　为 ScriptManager 控件添加属性

⑤ 在 body 的 form 标记中，可看到如下代码。

```
<asp:ScriptManager runat="server" ID="scriptManager">
    <Services> <asp:ServiceReference Path="GreetingsWebService.asmx" /> </Services>
    <Scripts> <asp:ScriptReference Path="callGreetingsClientJScript.js" /> </Scripts>
</asp:ScriptManager>
```

当然，可在源视图中直接输入上述 ScriptManager 控件的代码，这同样可完成上一步的操作。

该页面包含 ScriptManager 控件。Services 节的 ServiceReference 元素中的 path 属性引用先前生成的 GreetingsWebService.asmx 服务。Script 节的 ServiceReference 元素中的 path 属性引用 callGreetingsClientJScript.js 脚本。

ScriptManager 控件包括在 ASP.NET【AJAX 扩展】中，它用来处理页面上的所有组件以及页面局部更新，生成相关的客户端代理脚本以便能够在 JavaScript 中访问 Web Service，所有需要支持 ASP.NET AJAX 的 ASP.NET 页面上有且只能有一个 ScriptManager 控件。在 ScriptManager 控件中可以指定需要的脚本库，或者指定通过 JS 来调用的 Web Service，以及调用 AuthenticationService 和 ProfileService，还有页面错误处理等。

ASP.NET Web 服务通过 ScriptManager 控件向客户端脚本公开之后，ASP.NET 将为这些 Web 服务自动创建 JavaScript 代理类。可以通过调用 JavaScript 代理类的相应方法来调用 Web 服务方法。

⑥ 在 body 的 form 标记中，继续添加如下代码。

```
<span class="style1">记着要将 GreetingsWebService.cs 文件中的 Web 服务类的属性
   [System.Web.Script.Services.ScriptService]启用，即不要注释它。</span>
<hr />从客户端脚本调用：
<table>
<tr align="left">
    <td>Click to call the Greetings service:</td>
    <td><%--调用回调函数的控件是 HTML 控件--%>
        <input id="ButtonGreet" type="button"
        onclick ="OnClickGreetings(); return false;" value="Greetings" />
    </td>
</tr>
</table><hr />
```

这里将客户端 HTML 控件 Input(Button)的 onclick 属性和 callGreetingsClientJScript.js 客户端脚本文件中的函数 OnClickGreetings()绑定，在引发该 Input(Button)控件的 Click 事件时将调用并执行客户端脚本文件中的 OnClickGreetings 函数。

这里用的 id="ButtonGreet"按钮是【工具箱】的 HTML 组里的 Input(Button)，这是客户端控件，不同于服务器端控件 Button(【标准】组中的)。

⑦ 在浏览器中测试 Web 服务。打开浏览器，输入以下 URL。

```
http://localhost: 15605/callGreetingsService.aspx
```

⑧ 在显示的页中，单击 Greetings 按钮。这将生成对该 Web 服务的调用，这次调用将返回一条问候消息以及当前服务器日期和时间。

⑨ 切换到 callGreetingsService.aspx 的源视图，在 body 的 form 标记中，继续添加如下代码。

```
<hr />从客户端脚本调用：<br />输入用户名：
<asp:TextBox ID="TextBoxParameters"   runat="server"></asp:TextBox>
```

```
<asp:Button ID="ButtonGreetParam"  OnClientClick ="GreetingsParameters();
        return false;"  runat="server" Text="GreetingsParameters()" /><hr />
```

将下列 javascript 客户端脚本块添加到<head></head>标记后。

```
<script type ="text/javascript" >
  function  GreetingsParameters()
  {
     window.alert("now calling GreetingsParameters ()....");
     //获取文本框 TextBoxParameters 服务器控件的值
     var s=document.getElementById ("TextBoxParameters").value;
    window.alert(s);
    //调用客户端脚本文件 callGreetingsClientJScript.js 中的/函数 OnClickGreetingsParameters();
    OnClickGreetingsParameters(s);
  }
</script>
```

这里将服务器端控件 Button 的 OnClientClick 属性和 JavaScript 客户端脚本块中的函数 GreetingsParameters()绑定，在引发该 Button 控件的 Click 事件时将调用并执行客户端脚本中的 GreetingsParameters 函数。

⑩ 按 Ctrl+F5 组合键测试运行结果。

(6) 在服务器端使用 C#代码直接调用 Web 服务。

① 切换到 callGreetingsService.aspx 的【源】视图，在 body 的 form 标记中，继续添加如下代码。

```
<hr />从服务器端调用<br />请输入用户名
<asp:TextBox ID="TextBoxName" runat="server" ForeColor="#CCCCCC">猪八戒
  </asp:TextBox>
<asp:Button ID="ButtonServerSide" runat="server"   Text="从服务端调用 Greetings Web 服务"
  onclick="ButtonServerSide_Click" />
<asp:Label ID="LabelGreetings" runat="server" Text="Label"></asp:Label><hr/>
```

② 在 callGreetingsService.aspx.cs 文件中添加如下代码。

```
Samples.Aspnet.GreetingsWebService myGreetingsService =
        new Samples.Aspnet.GreetingsWebService();
protected void ButtonServerSide_Click(object sender, EventArgs e)
{
 //使用下面的代码可确定是否已为单个页启用  AJAX  功能。
 ScriptManager sm = ScriptManager.GetCurrent(Page);
 if (sm == null)
 {
   //确定是否已为本网页启用 ASP.NET AJAX 功能
   // ASP.NET AJAX functionality is not enabled for the page.
   string str = "alert('" + "ASP.NET AJAX functionality is not enabled for the page." + "')";
   Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "Warning", str, true);
 }
 else
 {
   // AJAX functionality is enabled for the page.
```

```
            LabelGreetings.Text =myGreetingsService.GreetingsParamters(TextBoxName.Text.ToString());
            string str = "alert('" + "ASP.NET AJAX functionality is    enabled for the page." + "')";
            Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "Warning", str, true);
        }
    }
```

③ 切换到 callGreetingsService.aspx 的源视图，按 Ctrl+F5 组合键，测试调用结果。

12.3　支持 AJAX 的 Web 服务的应用程序体系结构

典型的 AJAX 体系结构是相当容易理解的。如图 12-7 所示的是典型的 AJAX 体系结构图。其中有一个由应用程序特定服务组成的后端，通常只可调用 AJAX 脚本的外层，其下方是业务逻辑所在和发挥作用的系统中间层。服务与前端通过 HTTP 交换数据，使用多种格式传递参数和返回值。前端由运行于客户端上的 JavaScript 代码组成，在接收和处理完数据后，它面临着使用 HTML 和 JavaScript 构建图形用户界面的重大任务。对 JavaScript 的依赖是由于受浏览器结构的限制，只有当浏览器可以支持功能更加强大的编程功能时，这种情况才会改变。

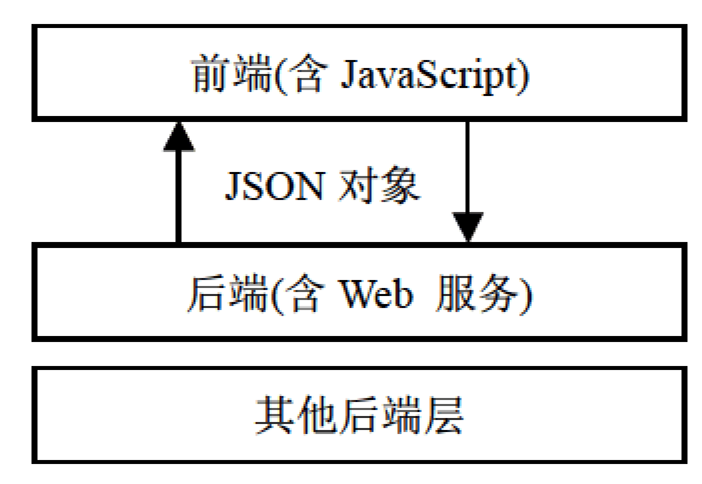


图 12-7　典型的 AJAX 体系结构

在 AJAX 中，服务表示驻留在应用程序域并向客户端脚本代码公开功能的一段代码。

最适合 AJAX 应用程序的服务主要 7 涉及向 Web 客户端公开数据和资源。它可以通过 HTTP 获得，并要求客户端使用 URL(也可以是 HTTP 头)访问数据和命令操作。客户端与服务进行交互使用的是 HTTP 动词，如 GET、POST、PUT 和 DELETE。换句话说，URL 代表一个资源，而 HTTP 动词描述了想对资源进行的操作。在这些交互中交换的数据以简单格式表示，如 JSON 和纯 XML，也可以聚合格式表示，如 RSS 和 ATOM。

这里用“AJAX 服务”表示通过 JavaScript 脚本服务方法实现的 AJAX 应用程序后端的服务。

12.3.1　AJAX 服务返回的数据

公开 AJAX 服务的唯一方式是通过 HTTP，几乎可以使用任何文本格式来包装请求和响应的主体。JavaScript Object Notation(JSON)是最常用的格式，但也可使用其他格式，如纯 XML 和原始文本。

JSON 是基于文本的格式，用于表示跨应用程序的各层的移动对象的状态。JSON 字符串通过常见的 eval 函数便可方便地赋给 JavaScript 对象。JSON 格式描述了对象，如下所示：

{"ID":"xunlei", "Company":"百度公司"}

该字符串表示一个对象有两个属性，即 ID 和 Company，以及它们各自的文本序列化值。如果对某个属性赋予非基本类型的值(比如自定义对象)，那么该值将递归地序列化为 JSON，如下所示。

```
{"ID":"xunlei","Company":"迅雷公司","Location": "{"City":"深圳", "Country":"中国"}",}
```

使用 eval 函数进行处理时，JSON 字符串将变成一个关联性数组(即一种名称/值的集合)，其中每个条目都有一个名称和值。如果 JSON 字符串用于代表一个自定义对象(比如 Customer)的状态，那么，必须负责确保客户端具有相应类的定义。换句话说，JavaScript 的 eval 函数只是将 JSON 字符串中的信息提取到一个通用容器。如果需要将此信息公开为一个自定义对象(比如 Customer 对象)，那么提供类定义并将数据载入到其中的任务就完全依靠设计者或设计者使用的框架来完成。

12.3.2　JSON 与 XML 之比较

过去 XML 被推崇为 Web 通用语言。现在，AJAX 正在改变着 Web，就数据表示而言，JSON 更受欢迎，而 XML 正在被推向角落。

JSON 比起 XML，稍显简单，更适合与 JavaScript 语言配合使用。Web 浏览器处理 JSON 比处理 XML 更容易。使用 JSON 时，不需要 XML 分析器之类的任何东西，为分析文本所需的一切都已经完全构建在 JavaScript 语言中。

通过 JSON 技术的使用，还能在体系结构方面以相对较低的成本赢得关键优势。可以按照对象无处不在的思路进行推理。在服务器上可定义一些实体，并用最喜爱的托管语言将它们实现为类。当某个 Web 服务方法需要返回任何类的一个实例时，该对象的状态被序列化为 JSON 并通过线路传送。客户端接收并处理 JSON 字符串，并将其内容载入一个数组或一种与服务器类有相同接口的镜像 JavaScript 对象。类的接口是从 JSON 流推断出来的。这样，服务和客户端页面的代码便使用一个实体的同一逻辑定义。

单纯从技术角度来说，AJAX 服务并不严格要求 JSON 实现为数据表示格式。使用 XML 也可实现同样的结果。

12.3.3　ASP.NET AJAX 中的服务

通过 ASP.NET AJAX 扩展实现脚本服务有两种方式——使用特殊类型的 ASP.NET Web 服务和通过页面方法。前一种情况下，只须设计和构建一个链接到 ASMX 资源的类。

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/TimeWebService.cs" Class="
IntroAjax.WebServices.TimeWebService" %>
```

该类可以选择从 WebService 类继承，并且必须用新的 ScriptService 属性加以修饰。

```
[ScriptService]
public class TimeWebService : System.Web.Services.WebService
{
  [WebMethod]
  public string GetServerTime()
  {
    string serverTime = String.Format("The current time is {0}.", DateTime.Now);
    return serverTime;
  }
}
```

每个可调用脚本的方法均声明为公共方法，并标记有通常的 WebMethod 属性。

页面方法只不过是在单个 ASP.NET 页面上下文中定义的公共、静态方法，每个方法都标记为 WebMethod。只能从宿主页面中调用它们。除了存储不同之外，对 Web 服务或页面方式的调用均由 ASP.NET AJAX 环境以相同方式进行处理。

AJAX 服务代表后端的一部分，从 WS-* Web 服务意义上讲，它们并不是公共 Web 服务，Web 服务都是通过 WSDL 脚本完整记录并可通过携带 SOAP 数据的 POST 命令进行访问。AJAX 服务是真正的本地服务，通常在调用它们的同一个应用程序中进行定义。然而，它们也可以在不同的 Web 应用程序甚至不同的网站上，前提是它们位于同一个域。

通过启用 ASP.NET AJAX 运行时以接受对服务的调用，ScriptService 属性扮演了一个关键角色。如果没有 ScriptService 属性，当用户试图进行调用时，服务器上就会引发异常。因为 ASP.NET 内部机制拒绝处理缺少 ScriptService 属性的 ASP.NET Web 服务进行的脚本调用。

默认情况下，调用 AJAX 服务方法的 JavaScript 代理类只能使用 HTTP POST 动词调用并返回 JSON 格式的数据。然而，可以通过给 Web 服务方法添加可选择的 ScriptMethod 属性，使它可以使用 HTTP GET 动词和 XML 格式。表 12-1 详细说明了 ScriptMethod 属性所支持的参数。返回到客户端的数据可以更改为 XML，甚至可以添加对 GET 请求的支持。但这种表面看起来无恶意的更改可能会为攻击者提供新的机会，且增加了跨站点调用该方法的可能性。下列代码显示了一个 Web 服务方法的定义。

```
[WebMethod]
[ScriptMethod]
public string GetServerTime()
{
    string serverTime = String.Format("The current time is {0}.", DateTime.Now);
    return serverTime;
}
```

表 12-1　ScriptMethod 属性的参数

参　数	说　明
ResponseFormat	指定是否将响应序列化为 JSON 或者 XML。默认为 JSON，但是，当方法的返回值是 XmlDocument 时，XML 格式会比较方便
UseHttpGet	表明是否可以使用 HTTP GET 动词调用 Web 服务方法。由于安全性原因，此项的默认设置为 false
XmlSerializeString	表明包括字符串在内的所有返回类型是否都序列化为 XML。默认为 false。当响应格式设置为 JSON 时，将忽略该属性的值

注意：

用 ScriptService 属性修饰的 AJAX 脚本服务仍可由能够访问 ASMX 服务端点且能处理 SOAP 协议的代码访问，如图 12-8 所示。

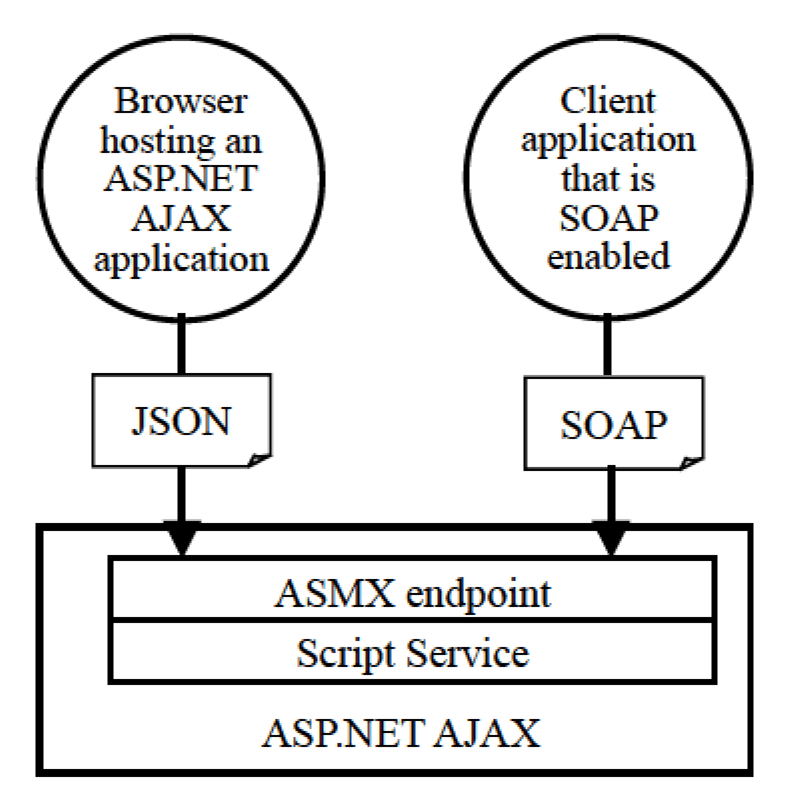


图 12-8　AJAX Web 服务采用 JSON 或 SOAP 通信格式

12.3.4　ASP.NET AJAX 服务和 SOAP

一旦创建了 ASP.NET AJAX 脚本服务，它就发布为 ASMX 资源。默认情况下，它是公共 URL，可以由 AJAX 客户端通过 JSON 格式使用，也可以由 SOAP 客户端及工具发现和使用。ASP.NET AJAX 服务默认情况下具有双重功能：JSON 和 SOAP。不过，可以选择禁用 SOAP 客户端及工具。只需在承载服务的 ASP.NET 应用程序的 Web.config 文件中输入下列配置设置即可关闭 SOAP 客户端。

```
<system.Web>
  <WebServices>
    <protocols>
      <clear />
    </protocols>
  </WebServices>
</system.Web>
```

这个简单的设置禁用了为 ASP.NET Web 服务定义的任何协议(特别是 SOAP)，并让服务仅回复 JSON 请求。

注意：

当这些设置开启时，无法再通过浏览器的地址栏调用 Web 服务来进行快速测试，也不能请求 WSDL 向 URL 中添加?wsdl 后缀。

为了在Web应用程序中启用ASP.NET AJAX，必须在Web.config文件中包含下列设置。

```
<httpHandlers>
  <remove verb="*" path="*.asmx"/>
  <add verb="*" path="*.asmx" validate="false"
     type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions,
     Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
  <add verb="*" path="*_AppService.axd" validate="false"
     type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions,
     Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
  <add verb="GET,HEAD" path="ScriptResource.axd"
```

```
        type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions,
        Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" validate="false"/>
</httpHandlers>
```

<remove>节会丢弃 ASMX 资源的默认 HTTP 处理程序，即通过 SOAP 处理请求的处理程序。<add>节点会添加一个新的 HTTP 处理程序，该处理程序主要检查每个传入的 ASMX 请求的内容类型，如果内容类型的头部设置为 application/json，则通过 JSON 处理该请求。否则，HTTP 处理程序假定该请求是基于 SOAP 的，并将其转发给 ASP.NET Web 服务处理程序。如果禁用了 SOAP 协议，则会拒绝该请求。

最后，不管表面如何，没有任何必要让 SOAP 介入 ASP.NET AJAX 服务。但对 SOAP 客户端的支持是得到保证的，除非在 Web.config 文件中明确禁用。

要使 ASP.NET AJAX 服务按预期正常工作，传入请求的内容类型 HTTP 头必须设置为 application/json。对于通过<script>标记实施的跨站点攻击来说，这还是极好的补救措施。

12.3.5　调用 AJAX 服务

(1) 添加调用 AJAX Web 服务的客户端 javascript 脚本文件，如以下代码。

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
  <Scripts> <asp:ScriptReference Path="~/ScriptPath/TimeServiceJScript.js" /> </Scripts>
</asp:ScriptManager>
```

(2) 添加 Web 服务客户端代理，如以下代码。

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
    <Services>   <asp:ServiceReference Path="TimeWebService.asmx" /> </Services>
</asp:ScriptManager>
```

这样，在客户端脚本中就即可直接调用 Web 服务的静态方法。

作为 ASP.NET Web 服务(Web services)而创建的 AJAX 脚本服务(AJAX script services)，必须用 JavaScript 代理类(JavaScript proxy classes)来调用。调用该 AJAX 脚本服务的 JavaScript 代理类，是由引用该 AJAX 脚本服务的客户端页面控件 ScriptManager 在页面下载后，由运行库自动生成并链接到客户端页面的。

JavaScript代理类具有与脚本服务相同的名称和许多附加属性。它的特征是有一组相同的方法，只不过采用了稍有扩展的签名，没有必要探究代理类的源代码。不过，如果想看一下它的结构，可从浏览器的地址栏中尝试调用URL：http://.../service.asmx/js。

浏览器将下载一个 JavaScript 文件，将其保存到本地磁盘，用“记事本”打开来详阅。

JavaScript 代理类继承 Sys.Net.WebServiceProxy 基类，它提供了进行 JSON 调用的基本功能。

表 12-2　JavaScript 代理类的属性

属　　性	说　　明
Path	指出底层 Web 服务的 URL
Timeout	指出在调用超时之前允许方法运行的毫秒数

(续表)

属　　性	说　　明
defaultSucceededCallback	指出调用成功时所调用的默认 JavaScript 回调函数
defaultFailedCallback	指出调用失败或超时情况下所调用的默认 JavaScript 回调函数(如果有)
defaultUserContext	指出要传递给成功回调或失败回调的默认 JavaScript 对象(如果有)

JavaScript 代理类的特征是具有表 12-2 中列出的属性。除了常规的一组参数之外，每个镜像的方法还有 3 个参数。这 3 个参数分别是：方法成功时所要调用的回调函数、方法失败或超时情况下所要调用的回调函数、传递给两个回调的上下文对象。

(3) 下面是从 ASP.NET AJAX 页面调用一个远程 AJAX 服务的一些示例代码。

```
function onclickgetServerTime()
{
  //window .alert ("calling onclickgetServerTime() ......");
  AjaxWebServices.TimeWebService.GetServerTime(onMethodComplete);
}
function onMethodComplete(results)
{   $get("Label1").innerHTML = results; }
```

在方法调用 GetServerTime(onMethodComplete)的实参中(无论结果如何)调用的成功回调函数 onMethodComplet 具有原型：function method(results, context, methodName)。

context 参数代表调用时指定的上下文对象。methodName 参数是一个设为服务方法名称的字符串。最后，对于成功调用时所调用的回调，results 参数是包含 JavaScript 版本的方法返回值的对象。对于失败的回调，此参数则代表 Sys.Net.WebServiceError 对象。

AJAX 完全是关于最广泛意义上的用户体验——连贯的感受、无闪烁更新、界面设计、资源聚合、实时数据等。但只能利用浏览器和它的一套可编程性功能，主要是浏览器的对象模型、DOM 实现、对 DHTML 扩展的支持、CSS、JavaScript 和插件。

JavaScript 是构建和操纵 UI 的主要工具。用户界面任务的典型模式要求客户端使用 JavaScript 调用远程服务、接收 JSON 数据或 XML 数据，然后重新整理页面以显示更改。

纯 AJAX 体系结构是基于客户端与服务器的松散耦合，即实质上彼此独立的两个世界通过 HTTP 线路就 JSON 交换消息连接了起来。在纯 AJAX 体系结构中有一个基于 Web 服务的后端和一个 JavaScript 驱动的前端。构建有效的 HTML UI 完全取决于设计者或设计者选择的控件库。不过，这种分离机制使 Web 开发人员能够继续采用 Silverlight 等新兴技术创建交互性更强的用户界面，而不受服务器平台的限制。

12.4　本章小结

本章全面讲述了 Web 服务技术。首先，介绍了支持 AJAX 的 Web Services 技术，内容包括 Web 服务的目标、影响、好处、主要特征、架构、类型，以及 JSON 和支持 AJAX

的 Web 服务的使用；接着介绍了创建和使用支持 AJAX 的 Web 服务；最后介绍了支持 AJAX 的 Web 服务的应用程序体系结构。通过本章的学习，读者应该能够掌握 Web 服务技术在应用程序中的使用。

12.5　思考和练习

一、单选题

1. 在使用 Web Service 前，要进行 Web Service 发现，常用的发现工具是(　)。

A、Wsdl.exe　B、Disco.exe　C、Ftp.exe　D、Ping.exe

2. 使用WebMethod属性的哪些子属性来表明XML Web服务类的方法可以使用状态(　)。

A、WebMothod(EnableSession = true)　B、WebMothod(EnableSession = false)

C、WebMothod(Enablecookie = true)　D、WebMothod(Enablecookie = false)

3. 要注册 XML Web 服务以供他人使用，需要注册的关键项是(　)。

A、业务、服务以及 tModel　B、企业联系人和服务

C、XML Web 服务的 URL 地址　D. 以上说法都不对

4. 作为一家咨询公司的技术主管，开发者正在寻找一种全球文档检索的服务，并且希望能在自己公司内部文档系统中集成这一功能。通过检索互联网上 Service Finder公司的全球网络服务列表，开发者找到了由 Document Searcher公司提供的搜索服务。咨询公司集成 Document Searcher公司提供的搜索服务时，采用何种协议可以了解服务的描述？(　)

A、RPC　B、UDDI　C、HTTP　D、SOAP

5. 以下关于发布与部署 Web Service 的说法哪个是对的？(　)。

A、发布与部署没有什么区别，两个仅是不同的定义

B、发布是将 Web Service 放到 IIS 上，部署是制作安装包

C、发布是将 Web Service 向外界公示，部署是将 Web Service 放到 IIS 上

D、发布会将 Web Service 的相关信息列入 UDDI 目录中方便查询，而部署仅实现了 Web Service 的物理可访问

二、多选题

1. 采用 SOAP 方法访问 Web Service 的优点是哪些？(　)

A、SOAP 协议比 HTTP 使用面更广　B、SOAP 协议支持的数据类型多

C、SOAP 协议支持丰富的功能扩展　D、SOAP 协议的访问流量比 HTTP 协议小

2. Web Method 支持的数据类型有哪些？(　)

A、所有简单类型　B、带有 in 或者 out 参数的类型

C、ref 类型的参数　D、不定长的参数列表

E、任何对象　F、DataSet

3. 用.net 开发 XML Web 服务，必须使用的命名空间有( )

A、System.Web.Services　B、System.Web.Services.Protocols

C、System.Threading　D、CreditCardValidtor

4. 以下关于 UDDI 哪些是错误的？( )

A、使用 Web Service 必须通过 UDDI

B、UDDI 能让你的 Web Service 获得更多的使用

C、UDDI 能提供一系列 Web Service 的最终访问点

D、UDDI 负责提供 WSDL 文件

E、若 UDDI 服务提供商无法访问， Web Service 也就无法访问

5. Web Service 需要以下哪些技术、协议的支持？( )

A、HTTP　B、SOAP　C、XML　D、TCP/IP E、UDDI

三、简答题

1. UDDI 商业注册中心所提供的信息从概念上分为三个部分：白页(White page)、黄页(Yellow page)、绿页(green page)，请你说明白页、黄页、绿页各代表什么意思？

2. 请说明使用 WebService 的优点以及局限？

第13章　在ASP.NET中使用XML

XML 被称为可扩展标记语言(Extensible Markup Language)，它提供了一种保存数据的格式，数据可以通过这种格式很容易地在不同的应用程序之间实现共享。

XML是专为Web设计的，.NET也把XML作为应用程序之间传递数据的一种主要方法。前面章节简单介绍了XML的基本概念，本章将详细介绍XML在ASP.NET中的使用。

本章的学习目标

- 了解 XML 的基本概念。
- 掌握如何利用 ADO.NET 访问 XML。
- 掌握如何利用 XML 类访问 XML。

13.1　XML 的基本概念

XML 是一种可以用来创建自己的标记的标记语言。它由万维网协会(W3C)创建，用来克服 HTML(即超文本标记语言，英文是 Hypertext Markup Language)，它是所有网页的基础)的局限。和 HTML 一样，XML 基于 SGML——标准通用标记语言(Standard Generalized Markup Language)。XML 是 SGML 上的一个子集，XML 包含了 SGML 很多特性，但是要比 SGML 简单得多。

XML 是一种类似于 HTML 的标记语言，但是 XML 不是 HTML 的替代品，XML 和 HTML 是两种不同用途的语言，其中最主要的区别是：XML 是专门用来描述文本的结构，而不是用于描述如何显示文本的，而 HTML 则是用来描述如何显示文本的。

XML 不像 HTML 那样提供了一组事先已经定义好的标记，而是提供了一个标准。利用这个标准，可以根据需要定义自己的新的标记。准确地说，XML 是一个元标记语言，它允许开发人员根据规则，制定各种各样的标记语言。

XML 是用来存放数据的，换句话来说，它可以作为微型数据库，这是最常见的数据型应用之一。可以利用相关的 XML API(MSXML DOM、JAVA DOM 等)对 XML 进行存取和查询。

总之，XML 是一种抽象的语言，它不如传统的程序语言那么具体。要深入地认识它，应该先从它的应用入手，选择一种需要的用途，然后再学习如何使用。

13.1.1　XML 的基本结构

下面就创建一个 XML 文档，用于保存学生信息。

【例 13-1】创建用于存放学生信息的 XML 文档。

(1) 启动 VS，新建名为 WebSite13，在【解决方案资源管理器】中，右击网站名，选择【添加】|【新建项】命令，在弹出的对话框中选择【数据】|【XML 文件】模板，更改名称为 studentinfo.xml，创建 XML 文档。

(2) 在 studentinfo.xml 中输入如下内容。

```
<?xml version="1.0" standalone="yes" ?>
<!--下面存放的是学生信息-->
<students>
    <student>
      <no>11</no>
      <name>Landy</name>
      <sex>女</sex>
      <birth>1989-6-01</birth>
      <address telephone="010-82982222">北京</address>
    </student>
    <student>
      <no>12</no>
      <name>陈晓旭</name>
      <sex>女</sex>
      <birth>1987-3-12</birth>
      <address telephone="010-63944000">北京</address>
    </student>
    …
</students>
```

第一行是 XML 版本说明，其作用是告诉浏览器或者其他处理程序：这个文档是 XML 文档。其中：version 表示文档遵守的 XML 规范的版本，如该例中的 version 是 1.0；standalone 表示文档内部包含文档类型定义 DTD。

注意：

版本说明必须是文档的第一行。

第二行是 XML 注释，其语法与 HTML 的注释一样。这并不是一种巧合，因为 XML 和 HTML 都是从 SGML 派生出来的。XML 文档注释不是强制性的，可以任意删除。

从第三行<students>标记一直到最后</students>标记，表示 XML 所包含的数据内容。

可以看出，XML 文档如同它的名字所指明的含义一样，是一个用标记语言创立的文档。它使用了自定义的各种标记来表示数据的含义。

13.1.2　标记、元素以及元素属性

标记是左尖括号(<)和右尖括号(>)之间的文本。有开始标记(如<name>)和结束标记(如</name>)。

元素是开始标记、结束标记以及位于二者之间的所有内容。在【例 13-1】中，<student>元素包含 5 个子元素：<no>、<name>、< sex>、<birth >和<address>。

属性是一个元素的开始标记中的名称–值对。在上面的示例中，telephone 是<address>元素的属性。

为了使一个 XML 文档结构完整，XML 必须遵守一定规则。常见的 XML 文档规则如下。

(1) 文档必须以 XML 版本声明开始。

(2) 含有数据的元素必须有起始标记和结束标记。每个起始标记必须以相应的结束标记结束。如果一个文档未能结束一个标记，浏览器将报告一个错误信息，并且不会以任何形式显示任何文档的内容。

(3) 不含数据并且仅使用一个标记的元素必须以/>结束。

(4) 文档只能包含一个能够包含全部其他元素的根元素，如<students>元素。

(5) 元素只能嵌套不能重叠。

(6) 属性值必须加引号。例如，<address telephone="021-30008655">中 telephone 属性的值不能写成< address telephone="021-30008655">。

13.1.3　XML 数据的显示

单独用 XML 是不能像页面那样显示数据的，必须使用某种格式化技术，如 CSS 或者 XSL，才能显示 XML 标记创建的文档。

前面已经讲到 XML 是将数据和格式分离的。XML 文档本身不知道如何来显示数据，必须有辅助文件来帮助实现。因为 XML 取消了所有标识，包括 font、color 和 p 等风格样式的定义标识，因此 XML 全部是采用类似 DHTML 中 CSS 的方法来定义文档样式。

XML 中用来设定显示风格样式的文件类型通常有 XSL 和 CSS 两种。

1. 使用 XSL 显示 XML

XSL 全称为 Extensible Stylesheet Language(可扩展样式表语言)，是用来设计 XML 文档显示样式的主要文件类型。它本身也是基于 XML 语言的。XSL 可以灵活地设置文档显示样式，XSL 可以将 XML 转化为其他的文档，如 HTML 文档。这样，就可以在浏览器上显示了。

下面用一个简单的 XSL 样式表来说明如何显示 XML 文档。限于篇幅，本书无法详述 XSL，有兴趣的读者可以参考其他书籍。

【例 13-2】用 XSL 样式表显示 XML 文档。

(1) 打开网站 WebSite13，在【解决方案资源管理器】中，右击网站名，选择【添加】|【新建项】命令，在弹出的对话框中选择【数据】|【XSLT 文件】模板，新建 XSL 样式表，默认名称为 XSLTFile.xslt。

(2) 在 XSLTFile.xslt 文档中添加如下内容。

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:msxsl="urn:schemas-microsoft-com:xslt" exclude-result-prefixes="msxsl">
      <xsl:template match="/">
         <html>
```

```
                <body>
                  <h2>学生信息</h2>
                  <table border="1">
                    <tr bgcolor="#ff6ec7">
                      <th align="left">学号</th>
                      <th align="left">姓名</th>
                      <th align="left">性别</th>
                      <th align="left">年龄</th>
                      <th align="left">住址</th>
                    </tr>
                    <xsl:for-each select="students/student">
                      <tr>
                        <td>   <xsl:value-of select="no" />   </td>
                        <td>   <xsl:value-of select="name" /> </td>
                        <td>   <xsl:value-of select="sex" /> </td>
                        <td>   <xsl:value-of select="birth" />   </td>
                        <td>   <xsl:value-of select="address" />   </td>
                      </tr>
                    </xsl:for-each>
                  </table>
                </body>
              </html>
            </xsl:template>
        </xsl:stylesheet>
```

以上代码中，前 3 行代码由系统自动生成，包括版本说明和命名空间引用等。<template>元素定义了一个 HTML 表格。表格的单元格将用于 XML 文档数据显示。

(3) 新建并打开 show2.aspx 网页，在页面上添加一个 XML 控件，该控件在工具箱的【标准】选项卡中。

(4) 设置 XML 控件的 DocumentSource 和 TransformSource 属性，将 DocumentSource 指向 studentinfo.xml 文档，TransformSource 指向 XSLTFile.xslt 文档。结果如下。

```
    <asp:Xml ID="Xml1" runat="server" DocumentSource="~/studentinfo.xml"
TransformSource="~/XSLTFile.xslt"></asp:Xml>
```

(5) 保存并按 Ctrl+F5 组合键运行，studentinfo.xml 中的内容就以表格的形式呈现在浏览器上，如图 13-1 所示。

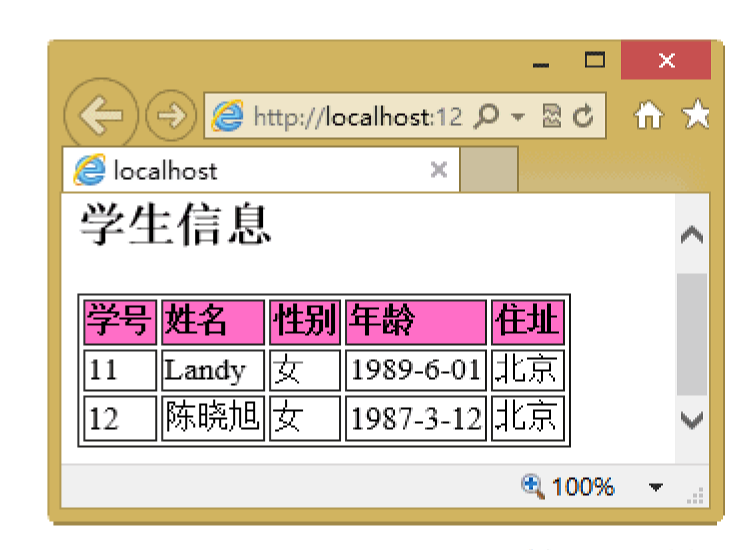


图 13-1　studentinfo.xml 的显示效果

2. 使用 CSS 显示 XML

CSS 大家很熟悉了，全称是 Cascading Style Sheets(层叠样式表)，是目前用来在浏览器上显示 XML 文档的主要方法。

下面用一个简单的 CSS 样式表来说明如何显示 XML 文档。

【例 13-3】用 CSS 样式表显示 XML 文档。

(1) 打开网站 WebSite13，在【解决方案资源管理器】中，右击网站名，选择【添加】

|【样式表】命令，新建 CSS 样式表 StyleSheet.css。

(2) 在 StyleSheet.css 样式表中添加如下代码。

```
name
{
    font-family: 宋体, Arial, Helvetica, sans-serif;
    font-size: large;
}
no,sex, birth, address
{
    font-family: 宋体, Arial, Helvetica, sans-serif;
    font-size: small;
}
```

(3) 打开 studentinfo.xml 文档，添加下面一行代码，建立与 StyleSheet.css 样式表的关联。

```
    …
    <?xml-stylesheet href="StyleSheet.css"
type="text/css" ?>
    <students>
    …
```

(4) 在浏览器上打开 studentinfo.xml，效果如图 13-2 所示。

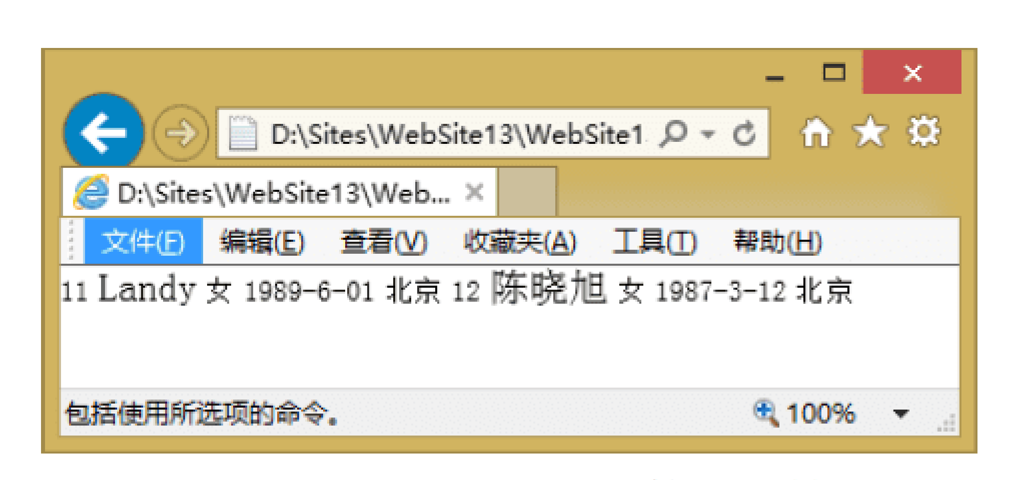


图 13-2　studentinfo.xml 的显示效果

13.2　使用 ADO.NET 访问 XML

在前面已经讨论了如何使用 ADO.NET 访问数据库的问题。数据库是进行数据存储和管理的一种习惯的方式。现在，XML 已逐步成为数据存储的一种新的方式，因此可以考虑将数据保存在 XML 文档中，并采用一定的方法对它进行管理。ADONET 提供了对 XML 数据访问的功能。

下面介绍如何使用 ADONET 访问 XML 数据。

13.2.1　将数据库数据转换成 XML

为了将数据库数据转换成 XML 文档，需要使用 DataSet 的 WriteXml 方法。WriteXml 方法只要指明要保存的 XML 文档的路径和文件名，就可以将 DataSet 中数据以 XML 的形式保存到 XML 文档中。

【例 13-4】将数据库数据转换成 XML 文档。

(1) 打开网站 WebSite13，在【解决方案资源管理器】中，右击网站名，选择【添加】|【新建项】命令，在弹出的对话框中选择【SQL Server 数据库】模板，更改名称为 MyDatabase.mdf，创建数据库。

(2) 在 MyDatabase.mdf 数据库中创建 student 表，并输入模拟数据。表的关系模式如下。

```
            Student(No,Name,Sex,Birthday,Address)
```

(3) 在 web.config 配置文件中，修改<connectionStrings/>标记如下。

```
    <connectionStrings>
    <add name="ConnectionString" connectionString="Data Source=.\SQLEXPRESS;
    AttachDbFilename=|DataDirectory|\ MyDatabase.mdf;Integrated Security=True;User
Instance=True"/>
    </connectionStrings>
```

(4) 新建网页 write_xml.aspx，在页面上添加一个 Label 控件。

(5) 给 write_xml.aspx 添加后台代码如下。

```
    //引用命名空间
    using System.Configuration;
    using System.Data;
    using System.Data.SqlClient;
    ……
    protected void Page_Load(object sender, EventArgs e)
    {
        //从 Web.Config 中取出数据库连接串
        string sqlconnstr =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;
        //创建连接对象
        SqlConnection sqlconn = new SqlConnection(sqlconnstr);
        //创建 DataSet 对象
        DataSet ds = new DataSet();
        //打开连接
        sqlconn.Open();
        //创建适配器对象
        SqlDataAdapter sqld = new SqlDataAdapter("select no,name,sex, CONVERT(char(10), birth,20)
AS birth,address from student", sqlconn);
        //利用适配器方法添加数据给 DataSet
        sqld.Fill(ds, "student");
        //将 DataSet 数据写成 XML 文本
        ds.WriteXml(Server.MapPath("students.xml"));
        sqlconn.Close();
        Label1.Text = "写入成功";
    }
```

(6) 运行程序，打开程序运行时保存的 students.xml 文件，内容如下。

```
    <?xml version="1.0" standalone="yes"?>
    <NewDataSet>
      <student>
        <no>1</no>
        <name>段洋</name>
        <sex>男</sex>
        <birth>1988-05-21</birth>
        <address>北京</address>
      </student>
      <student>
```

```
        <no>2</no>
        <name>李为明</name>
        <sex>男</sex>
        <birth>1982-05-01</birth >
        <address>北京</address>
    </student>
</NewDataSet>
```

可以看出，这个文档保存了 student 数据表中所有的数据。其中使用<NewDataSet>作为根结点标记，<student>作为每个记录的标记(student 是 sqld.Fill(ds, "student")语句中使用的名字)，另外，每个字段的名字作为数据元素的标记名。

13.2.2　读取 XML 文档

使用 DataSet 的 ReadXml 方法可以读取所有 XML 文档数据。下面通过【例 13-5】介绍读取 XML 文档的方法。

【例 13-5】读取 XML 文档。

(1) 打开网站 WebSite13，新建网页 read_xml.aspx，在页面上添加 1 个 GridView 控件。

(2) 给 read _xml.aspx 添加后台代码如下。

```
using System.Data;
……
protected void Page_Load(object sender, EventArgs e)
{
        DataSet ds = new DataSet();
        //读取 XML 文本数据到 DataSet 数据集
        ds.ReadXml(Server.MapPath("students.xml"));
        //绑定数据源
        GridView1.DataSource = ds.Tables[0].DefaultView;
        GridView1.DataBind();
}
```

(3) 程序执行结果如图 13-3 所示。

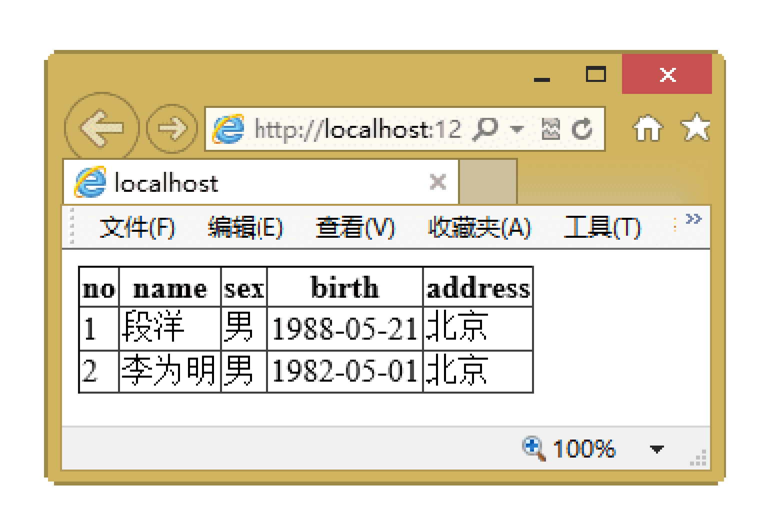


图 13-3　read_xml.aspx 的运行效果

13.2.3　编辑 XML 文档

编辑 XML 文档的方法也很简单，只要使用 DataSet 的 ReadXml 方法把 XML 数据读到 DataSet 中之后，修改相应的记录值，再使用 DataSet 的 WriteXml 方法保存 XML 文档就可以了。

【例 13-6】编辑 XML 文档。

(1) 打开网站 WebSite13，新建网页 edit_xml.aspx，在页面上添加 1 个 GridView 控件。

(2) 给 edit _xml.aspx 添加后台代码如下。

```
using System.Data;
……
protected void Page_Load(object sender, EventArgs e)
```

```
{
    //建立 DataSet 对象
    DataSet ds = new DataSet();
    ds.ReadXml(Server.MapPath("students.xml"));
    //建立 DataTable 对象
    DataTable dtable;
    //建立 DataRowCollection 对象
    DataRowCollection coldrow;
    //建立 DataRow 对象
    DataRow drow;
    //将数据表 tabstudent 的数据复制到 DataTable 对象
    dtable = ds.Tables[0];
    //用 DataRowCollection 对象获取这个数据表的所有数据行
    coldrow = dtable.Rows;
    //修改操作，逐行遍历，取出各行的数据
    for (int inti = 0; inti < coldrow.Count; inti++)
    {
        drow = coldrow[inti];
        //给每位学生姓名后加上字母 A
        drow[1] = drow[1] + "A";
    }
    //将 DataSet 数据写成 XML 文本
    ds.WriteXml(Server.MapPath("students.xml"));
    //绑定数据源
    GridView1.DataSource = ds.Tables[0].DefaultView;
    GridView1.DataBind();
}
```

(3) 程序执行结果如图 13-4 所示。

经过修改后，每个学生姓名增加了大写字母 A。可以看到，使用 ADO.NET 编辑 XML 文档，实际上就是对 DataSet 数据集数据的编辑，使用非常简单。

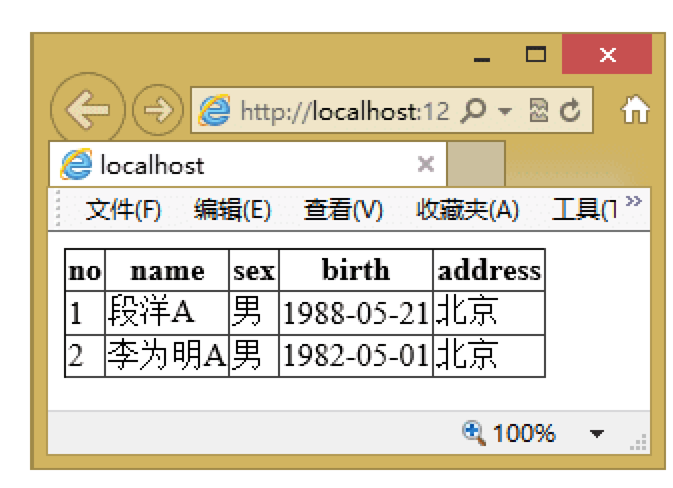


图 13-4　edit _xml.aspx 的运行效果

13.2.4　将 XML 写入数据库

XML 文本内容写入数据库与数据库的数据转换为 XML 是个相反的过程，需要用到 DataAdapter 的 Update 方法。下面列举一个简单的例子，将 studentinfo.xml 文档中仅有的学号为 11 和 12 的两位学生的数据，写入到数据库的 student 表中。

【例 13-7】将 XML 写入数据库。

(1) 打开网站 WebSite13，新建网页 xml_table.aspx，在页面上添加 1 个 GridView 控件。

(2) 给 xml_table.aspx 添加后台代码如下。

```
//引用命名空间
using System.Data.SqlClient;
using System.Data;
using System.Configuration;
……
```

```
    protected void Page_Load(object sender, EventArgs e)
    {
        //从 Web.Config 中取出数据库连接串
        string sqlconnstr =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;
        //创建连接对象
        SqlConnection sqlconn = new SqlConnection(sqlconnstr);
        //创建 DataSet 对象
        DataSet ds = new DataSet();
        //打开连接
        sqlconn.Open();
        //创建适配器对象
        SqlDataAdapter sqld = new SqlDataAdapter("select * from student", sqlconn);
        //利用适配器方法添加数据给 DataSet
        sqld.Fill(ds, "student");
        DataTable dt = ds.Tables["student"];
        //读取 XML
        dt.ReadXml(Server.MapPath("studentinfo.xml"));
        //自动生成提交语句
        SqlCommandBuilder objcb = new SqlCommandBuilder(sqld);
        //提交数据库
        sqld.Update(ds, "student");
        //绑定数据源
        GridView1.DataSource = ds.Tables["student"].DefaultView;
        GridView1.DataBind();
    }
```

(3) 按 Ctrl+F5 组合键执行程序，观看程序运行效果。可以看到，新增了 11 和 12 号两位学生。

13.2.5　将 XML 数据转换为字符串

前面讲到的所有方法都是使用 DataSet 来进行数据处理的。在实际工作中，如果希望进行 XML 数据传输，那么把 XML 数据读出之后形成字符串，即把数据当成字符串法进行处理。例如，可以把数据写在一个普通的 E-mail 中发送给其他的用户，对方就可以采用普通字符串处理的方法得到数据。为了能够完成上述功能，DataSet 还提供了将 XML 数据转换为字符串的方法 GetXml。

【例 13-8】将 XML 数据转换为字符串。

(1) 打开网站 WebSite13，新建网页 Trans_xml.aspx，在页面上添加 1 个 Label 控件。

(2) 给 Trans_xml.aspx 添加后台代码如下。

```
    using System.Data;
    ……
    protected void Page_Load(object sender, EventArgs e)
    {
        DataSet ds = new DataSet();
        //读取 XML 文本数据到 DataSet 数据集
```

```
        ds.ReadXml(Server.MapPath("students.xml"));
        //将 DataSet 的数据转换为字符串
        Label1.Text = ds.GetXml();
    }
```

(3) 运行程序，执行效果如图 13-5 所示。

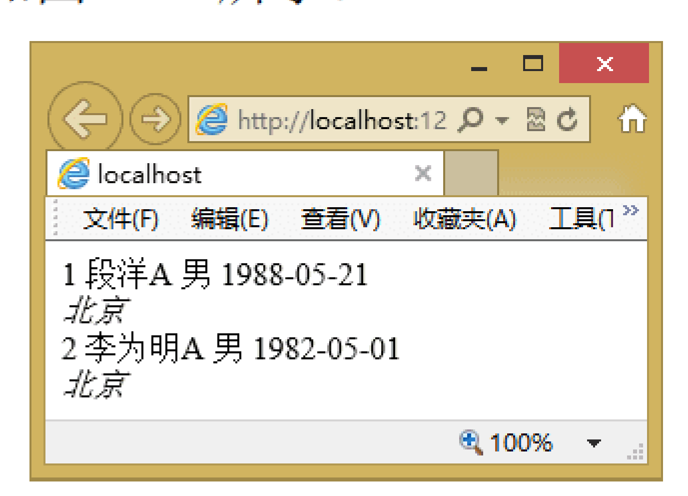


图 13-5　Trans_xml.aspx 的执行效果

13.3　使用.NET 的 XML 类访问 XML

ASP.NET 通过 System.Xml 命名空间为开发人员提供了操作 XML 的所有功能。该命名空间包括许多类，常用类如表 13-1 所示。

表 13-1　System.Xml 命名空间常用类

类	说　明
XmlReader	抽象的读取器类，提供快速、没有缓存的 XML 数据。XmlReader 是只向前的，类似于 SAX 分析器
XmlWriter	抽象类的写入器，以流或文件的格式提供快速、没有缓存的 XML 数据
XmlTextReader	扩展 XmlReader，提供访问 XML 数据的快速只向前流
XmlTextWriter	扩展 XmlWriter，快速生成只向前的 XML 流
XmlNode	抽象类，表示 XML 文档中一个节点的类。XML 命名空间中几个类的基类
XmlDocument	扩展 XmlNode，W3C DOM 的实现，给出 XML 文档在内存中的树形表示，可以浏览和编辑它们
XmlDataDocument	扩展 XmlDocument，即从 XML 数据中加载的文档，或从 ADO.NET DataSet 的关系数据中加载的文档，允许把 XML 和关系数据混合在同一个视图中
XmlResolver	抽象类，分析基于 XML 的外部资源，如 DTD 和模式引用，也可以用于处理和元素
XmlUrlResolver	扩展 XmlResolver，用 URI(Uniform Resource Identifier)解析外部资源

有关 XML 类编程技术内容较多，本书在此仅通过下面的【例 13-9】和【例 13-10】来简单说明 XML 类编程的过程。

【例 13-9】将数据库 student 表中男同学的数据保存到 XML 文档中。

(1) 打开网站 WebSite13，新建网页 write_xml2.aspx，在页面上添加 1 个 Label 控件。

(2) 给 write_xml2.aspx 添加后台代码如下。

```
    //引用命名空间
    using System.Data.SqlClient;
    using System.Xml;
    using System.Configuration;
    using System.Data;
    …
    protected void Page_Load(object sender, EventArgs e)
    {
        //从 Web.Config 中取出数据库连接串
        string sqlconnstr =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;
        //创建连接对象
        SqlConnection sqlconn = new SqlConnection(sqlconnstr);
        //创建 DataSet 对象
        DataSet ds = new DataSet();
        //打开连接
        sqlconn.Open();
        //创建适配器对象
        SqlDataAdapter sqld = new SqlDataAdapter("select * from student", sqlconn);
        //利用适配器方法添加数据给 DataSet
        sqld.Fill(ds, "student");
        DataTable dt = ds.Tables["student"];
        //创建 XML 文档
        XmlDocument xmldoc = new XmlDocument();
        //写入版本说明
        XmlDeclaration xmldecl = xmldoc.CreateXmlDeclaration("1.0", "utf-8", "yes");
        xmldoc.AppendChild(xmldecl);
        //写入根结点
        XmlElement students = xmldoc.CreateElement("students");
        xmldoc.AppendChild(students);
        //对每一行循环
        for (int i = 0; i < dt.Rows.Count; i++)
        {
            if (dt.Rows[i][2].ToString()=="男")
            {
                XmlElement student = xmldoc.CreateElement("student");
                //循环每一列
                for (int j = 0; j < dt.Columns.Count; j++)
                {
                    XmlElement colname = xmldoc.CreateElement(dt.Columns[j].ColumnName);
                    colname.InnerText = dt.Rows[i][j].ToString();
                    student.AppendChild(colname);
                }
                students.AppendChild(student);
            }
        }
        //将 xmldoc 数据写成 XML 文本
        xmldoc.Save(Server.MapPath("student_men.xml"));
        sqlconn.Close();
        Label1.Text = "写入成功";
    }
```

(3) 运行程序，打开程序运行时保存的 student_men.xml 文件，可以看出 XML 文档中只保存了男生的数据。

【例 13-10】将 studentinfo.xml 文档中男生数据写到 student 数据表中。

(1) 打开网站 WebSite13，新建网页 xml_table2.aspx，在页面上添加 1 个 GridView 控件。

(2) 给 xml_table2 .aspx 添加后台代码如下。

```
//引用命名空间
using System.Data.SqlClient;
using System.Xml;
…
protected void Page_Load(object sender, EventArgs e)
{
    //从 Web.Config 中取出数据库连接串
    string sqlconnstr =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;
    //创建连接对象
    SqlConnection sqlconn = new SqlConnection(sqlconnstr);
    //创建 DataSet 对象
    DataSet ds = new DataSet();
    //打开连接
    sqlconn.Open();
    //创建适配器对象
    SqlDataAdapter sqld = new SqlDataAdapter("select * from student", sqlconn);
    //利用适配器方法添加数据给 DataSet
    sqld.Fill(ds, "student");
    DataTable dt = ds.Tables["student"];
    //创建 XML 文档
    XmlDocument xmldoc = new XmlDocument();
    //读取 xml
    xmldoc.Load(Server.MapPath("studentinfo.xml"));
    //得到整个元素
    XmlElement xmle = xmldoc.DocumentElement;
    //遍历每个元素
    for (int i = 0; i < xmle.ChildNodes.Count; i++)
    {
        XmlNode xmln = xmle.ChildNodes[i];
        if (xmln.ChildNodes[2].InnerText == "男")
        {
            DataRow newrow = dt.NewRow();
            newrow["no"] = xmln.ChildNodes[0].InnerText;
            newrow["name"] = xmln.ChildNodes[1].InnerText;
            newrow["sex"] = xmln.ChildNodes[2].InnerText;
            newrow["birth"] = xmln.ChildNodes[3].InnerText;
            newrow["address"] = xmln.ChildNodes[4].InnerText;
            dt.Rows.Add(newrow);
        }
    }
    //自动生成提交语句
    SqlCommandBuilder objcb = new SqlCommandBuilder(sqld);
```

```
        //提交数据库
        sqld.Update(ds, "student");
        //绑定数据
        GridView1.DataSource = ds.Tables["student"].DefaultView;
        GridView1.DataBind();
    }
```

(3) 按 Ctrl+F5 组合键运行程序，可以看到 studentinfo.xml 文档中 11 号男生数据被写到了数据库的 student 表中。

13.4　XML 的应用和发展前景

虽然人们对 XML 的某些技术标准尚有争议，但是人们已经普遍认识到 XML 的作用和巨大潜力。下面对 XML 应用和发展前景作一个简要介绍。

13.4.1　XML 的应用

作为互联网的新技术，XML 的应用非常广泛，可以说 XML 已经渗透到了互联网的各个角落。考察现在的 XML 应用，可以大致将它分为以下几个方面。

1. 数据交换

利用 XML 在应用程序之间作数据交换已不是什么秘密了，毫无疑问应被列为第一位。那么为什么 XML 在这个领域里的地位这么重要呢？原因就是 XML 使用元素和属性来描述数据。在数据传送过程中，XML 始终保留了诸如父、子关系这样的数据结构。几个应用程序可以共享和解析同一个 XML 文件，不必使用传统的字符串解析或拆解过程。

相反，普通文件不对每个数据段做描述，也不保留数据关系结构。使用 XML 做数据交换可以使应用程序更具有弹性，因为可以用位置(与普通文件一样)或用元素(从数据库)来存取 XML 数据。

2. Web 服务

Web 服务是最令人激动的革命之一，它让使用不同系统和不同编程语言的人们能够相互交流和分享数据。其基础在于 Web 服务器用 XML 在系统之间交换数据。交换数据通常用 XML 标记，能使协议取得规范一致，如在简单对象处理协议(Simple Object Access Protocol，即 SOAP)平台上。

SOAP 可以在用不同编程语言构造的对象之间传递消息。这意味着一个 C#对象能够与一个 Java 对象进行通信。这种通讯甚至可以发生在运行于不同操作系统上的对象之间。DCOM，CORBA 或 Java RMI 只能在紧密耦合的对象之间传递消息，SOAP 则可在松耦合对象之间传递消息。

3. 内容管理

XML 只用元素和属性来描述数据，而不提供数据的显示方法。这样，XML 就提供了一个优秀的方法来标记独立于平台和语言的内容。

使用像 XSLT 这样的语言能够轻易地将 XML 文件转换成各种格式文件，如 HTML、WML、PDF、EDI 等。XML 具有的能够运行于不同系统平台之间和转换成不同格式目标文件的能力使得它成为内容管理应用系统中的优秀选择。

4. Web 集成

现在有越来越多的设备支持 XML 了。使得 Web 开发商可以在个人电子助理和浏览器之间用 XML 来传递数据。

为什么将 XML 文本直接送进这样的设备去呢？这样做的目的是让用户更多地自己掌握数据显示方式，更能体验到实践的快乐。常规的客户/服务(C/S)方式为了获得数据排序或更换显示格式，必须向服务器发出申请；而 XML 则可以直接处理数据，不必经过向服务器申请查询，返回结果这样的双向“旅程”，同时设备也不需要配制数据库，甚至还可以对设备上的 XML 文件进行修改并将结果返回给服务器。

5. 配置文件

许多应用都将配置数据存储在各种文件里，如.INI 文件。虽然这样的文件格式已经使用多年并一直很好用，但是 XML 还是以更为优秀的方式为应用程序标记配制数据。使用.NET 里的类，如 XmlDocument 和 XmlTextReader，将配置数据标记为 XML 格式，能使其更具可读性，并能方便地集成到应用系统中去。使用 XML 配置文件的应用程序能够方便地处理所需数据，不用像其他应用那样要经过重新编译才能修改和维护应用系统。

13.4.2　XML 的发展前景

XML 自推出以来，尤其是在 1998 年 2 月成为 W3C 推荐标准以来，受到了广泛地支持。各大软件厂商如 IBM、Microsoft、Oracle、Sun 等都积极支持并参与 XML 的研究和产品化工作，先后推出了支持 XML 的产品或者改造原有的产品以支持 XML，W3C 也一直致力于完善 XML 的整个理论体系。

XML 虽然获得了极大的支持，但是它还有一段路要走。首先，XML 的规则还有许多技术细节没有解决。其次，现在虽然出现了一些 XML 工具和应用，但是其市场反应还有待进一步观察。另外如何让更多的人迅速学会使用 XML，并利用它进行开发，进而促进 XML 的应用也是一个问题。XML 的出现和迅猛发展并不意味着 HTML 即将退出互联网舞台。由于 HTML 的易学易用和非常多的工具支持，HTML 将在较长的时间里继续在 Web 舞台上充当主角。但如果用户想超越 HTML 的范围，XML 将是最佳的选择。

另外，由于 XML 是用于自定义标记的元语言，任何个人、公司和组织都可以利用它来定义自己的标记语言。这虽然是 XML 的魅力和灵活性之所在，但同时也是 XML 的最大问题之所在。如果每个人、公司和组织都定义了自己的标记语言，它们之间的通信就会出

现困难。因此在一些领域先后出现了一些标准化组织，它们的任务就是规范本领域的标记语言，形成统一的标准，使得在本领域内的通信成为可能。但在标准推出并得到广泛认可之前，各自为政的局面将继续下去。更糟糕的是，由于对应用的理解不一致和商业利益等原因，同一个领域也许还有多个标准化组织，它们形成的标记语言并不完全兼容，使得采取不同标准的计算机仍然难以通信。

无论如何，XML 的出现使互联网跨入了一个新的阶段，它已成为因特网领域中一个重要的数据交换标准和开发平台。没有 XML 就没有 Web 服务，也就没有今天构建应用软件的轰轰烈烈的 SOA(Service Oriented Architecture)。XML 的诞生已经而且将继续促使全新种类的基础架构和应用程序的产生，而这些新的基础架构和应用程序又将需要新的软件和硬件工具。可以预测，无论是在软件还是硬件上，XML 都将开辟一系列的新市场，促成互联网上新的革命。

13.5　本章小结

本章全面介绍了 XML 技术在 ASP.NET 应用程序中的使用方法。首先介绍了 XML 的基本概念，包括 XML 的基本结构，标记、元素及其属性，以及通过 XSL 和 CSS 显示 XML 文件内容的方法；然后介绍了使用 ADO.NET 访问 XML 文件的方法，内容包括将数据库数据转换成 XML 文档、读取 XML 文档、编辑 XML 文档、将 XML 写入数据库、将 XML 数据转换为字符串的方法等；接着介绍了使用.NET 的 XML 类访问 XML 文档；最后简单介绍了 XML 的应用和发展前景。通过本章的学习，读者能够掌握 XML 技术在 ASP.NET 应用程序中的使用方法。

13.6　思考和练习

1. XML 文档的结构如何？
2. XML 文档的规则有哪些？
3. XML 文档的内容有哪几种呈现方法？
4. 如何通过 ADO.NET 访问 XML？
5. XML 的应用和发展前景如何？

第14章　网上书店

近些年，电子商务一直是互联网的主流。随着互联网和移动网络的融合，更是壮大了电子商务行业，越来越多的商家在网上开起了在线商店。这种网上购物系统具有强大的交互功能，可使商家和用户方便地传递信息，完成电子贸易，它向消费者展示出一种新颖的购物理念。

本章将介绍一个结构清晰的网上购物实例——网上书店，该实例可以帮助读者掌握用ASP.NET 开发网上购物系统的方法。

本章学习目标：

- 网上书店系统的系统设计与数据库设计。
- 用户管理模块。
- 图书管理模块。
- 购物车模块。
- 订单管理模块。
- 后台管理模块。

14.1　项目概述

只有了解一个系统的整体设计，才能对系统中的各种模块及应用有所了解。本节将对网上书店项目的设计方案、数据库设计、模块划分进行一个总体上的介绍，使得读者对项目有个总体的印象。

14.1.1　设计方案

网上书店系统包括用户注册和登录、管理图书类别、管理图书信息、搜索图书、管理购物车、管理订单、管理用户信息、管理销售情况等功能，这些功能的具体描述如下。

- 用户注册和登录：为用户提供注册、登录、找回丢失密码、修改个人信息等功能。
- 图书信息查询及管理：对信息进行灵活分类、存储，方便用户迅速从少则几万，多则几十万甚至上百万种图书中找出自己所需要的图书。
- 购物车管理：用于存储用户选择好的图书，完成购物车后可以自动生成订单以供管理者进行处理。
- 订单管理：为用户提供订单查询功能，同时为管理者提供订单查询及处理功能。
- 后台管理：为管理者提供用户信息查询和销售情况查询等功能。

为实现以上功能，本系统分为 4 层，它们分别为表示层(由 ASP.NET Web 窗体组成，主要用于显示信息和与用户交互)、商务逻辑层(用户抽象表示层功能，为表示层提供服务)、数据访问层(为商务逻辑层提供访问数据库系统的接口)和系统数据库。

- 表示层页面包括 3 类：用于处理相关图书信息的 Book*.aspx 页面、用于处理用户信息的 User*.aspx 页面、用于实现管理的 Manage*.aspx 页面。
- 商务逻辑层包括 Book.cs、User.cs、ShopCart.cs、Order.cs、Admin.cs 文件，分别封装了为实现对图书、用户、购物车、订单、管理进行操作而需的方法。该层可以直接访问数据访问层，并为表示层提供了访问接口，起到了表示层和数据访问层的桥梁作用。
- 数据访问层包含 SqlAccess.cs 文件，封装了对 SQL Server 数据库中数据的读取和对存储过程的操作。为了给服务器减负，本例特地选用 SQL Server 数据库。

本系统的层次结构如图 14-1 所示。

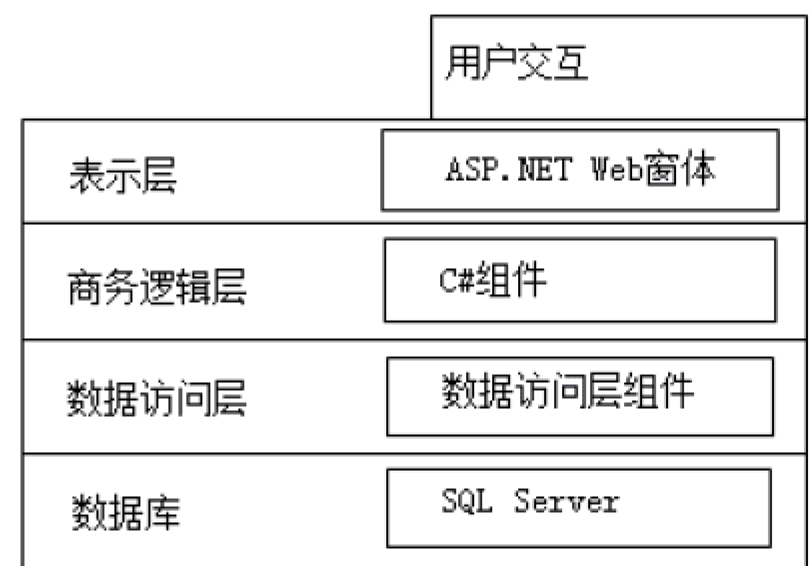


图 14-1　系统层次结构图

14.1.2　数据库设计

本系统是一个典型的电子商务系统，数据库中包含 6 张表，分别如下。

1. 图书类别表 BookCategory

网站的图书种类繁多，用图书类别表存储网站图书种类信息，可以简化图书管理，该表字段说明如表 14-1 所示。

表 14-1　图书类别表 BookCategory

字段名	类型	长度	允许空	键值	备注
CategoryId	Int	4	否	是	图书类别 ID 号
CategoryName	nvarchar	50	是	否	图书类别名称

2. 图书信息表 BookInfo

图书信息表存储网站图书具体信息，该表引用 BookCategory 表的 CategoryId 字段作为外键，具体字段说明如表 14-2 所示。

表 14-2　图书信息表 BookInfo

字段名	类型	长度	允许空	键值	备注
BookId	Int	4	否	是	图书 ID 号
BookName	nvarchar	50	是	否	图书名称
Cover	binary	50	是	否	图书封面
CategoryId	int	4	是	否	图书类别 ID 号

(续表)

字段名	类型	长度	允许空	键值	备注
Author	nvarchar	50	是	否	图书作者
Translator	nvarchar	50	是	否	图书译者
Press	nvarchar	50	是	否	图书出版社
Price	decimal	9	是	否	图书价格
Introduction	nvarchar	200	是	否	图书介绍

3. 订单表 Order

订单表用来存储用户订单，引用 UserInfo 表的 UserId 字段作为外键，具体字段内容如表 14-3 所示。

表 14-3　订单表 Order

字段名	类型	长度	允许空	键值	备注
OrderId	int	4	否	是	订单 ID 号
UserId	int	4	是	否	用户 ID 号
OrderDate	datetime	8	是	否	订单日期
Status	nvarchar	50	是	否	订单状态

4. 订单信息表 OrderInfo

订单信息表包含具体订单信息，引用 BookInfo 表的 BookId 作为外键，具体字段说明如表 14-4 所示。

表 14-4　订单信息表 OrderInfo

字段名	类型	长度	允许空	键值	备注
OrderId	int	4	否	是	订单 ID 号
BookId	int	4	否	是	图书 ID 号
Quantity	int	4	是	否	图书数量
UnitCost	decimal	9	是	否	图书价格

5. 购物车表 ShopCart

购物车表主要存储网站用户购物车信息，该表引用 BookInfo 表的 BookId 作为外键，其具体字段如表 14-5 所示。

表 14-5　购物车表 ShopCart

字段名	类型	长度	允许空	键值	备注
Id	int	4	否	是	ID
CartId	nvarchar	50	是	否	购物车 ID 号
BookId	int	4	是	否	图书 ID 号
Quantity	int	4	是	否	图书数量
CreateDate	datetime	6	是	否	创建日期

6. 用户信息表 UserInfo

用户信息表主要用于存储网站用户信息，字段说明如表 14-6 所示。

表 14-6　用户信息表 UserInfo

字段名	类型	长度	允许空	键值	备注
UserId	int	4	否	是	用户 ID 号
UserName	nvarchar	50	是	否	用户名称
UserPwd	nvarchar	50	是	否	用户密码
UserRole	nvarchar	50	是	否	用户角色
UserRealName	nvarchar	50	是	否	用户真实姓名
Question	nvarchar	50	是	否	用户提问
Answer	nvarchar	50	是	否	用户答案
Email	nvarchar	50	是	否	邮箱地址

各数据表如图 14-2 所示。

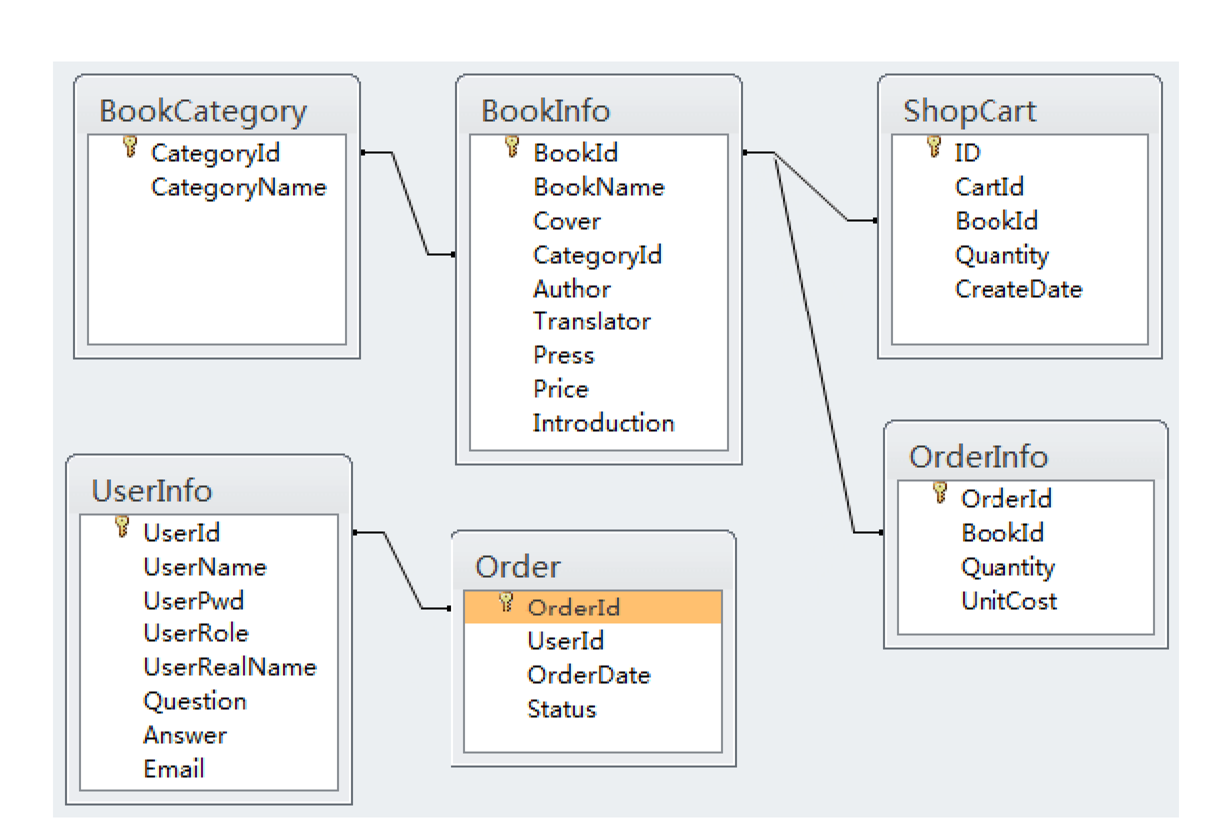


图 14-2　数据图

14.1.3　模块划分

该系统的整体模块组织图如图 14-3 所示。

该系统为普通用户和管理员提供了不同的操作界面，普通用户的操作界面如图 14-4 所示。管理员的操作界面如图 14-5 所示。

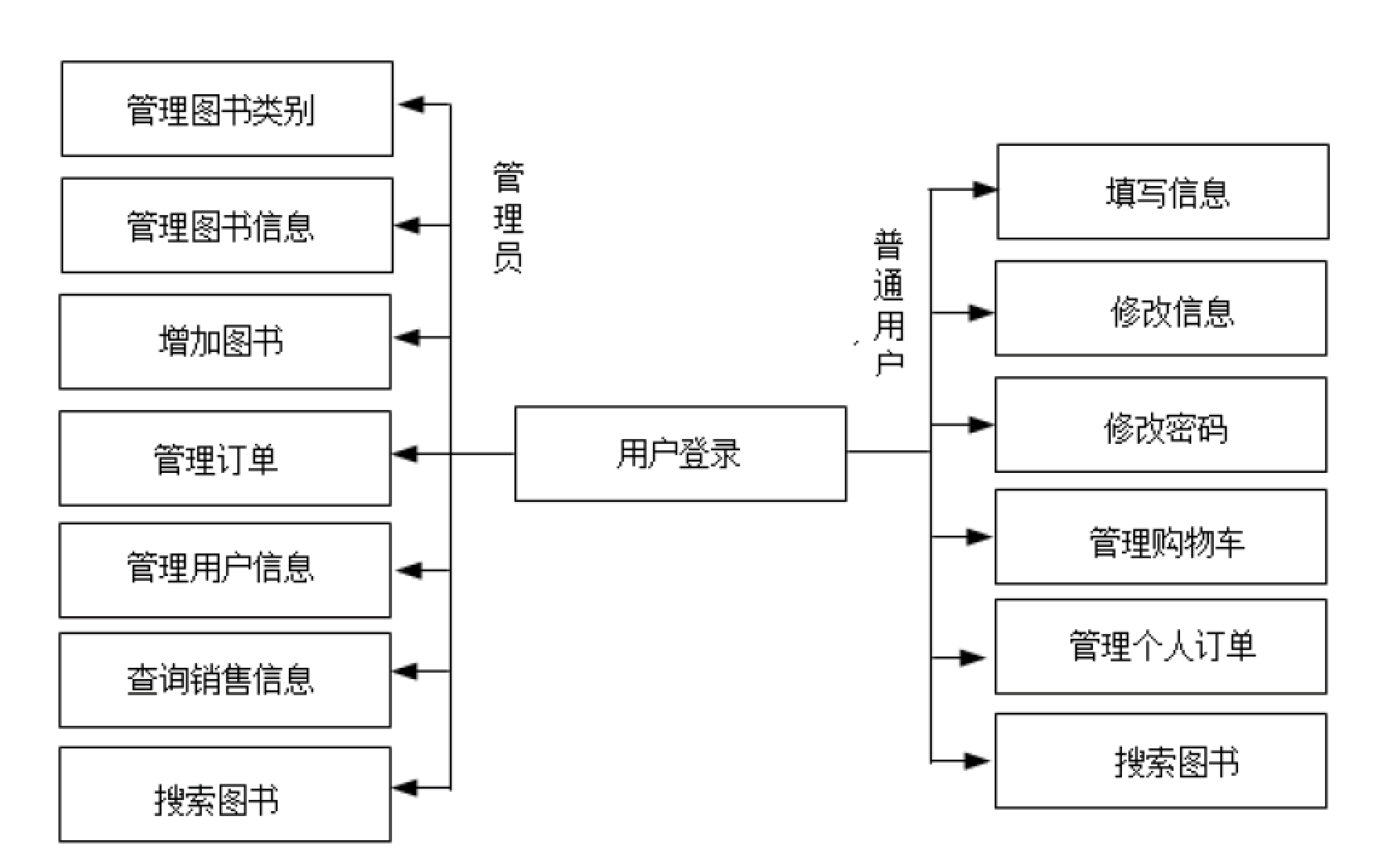


图 14-3　系统模块组织图



图 14-4　普通用户界面



图 14-5　管理员界面

普通用户界面划分为修改信息、修改密码、我的购物车、我的订单、搜索等模块；管理员界面划分为修改信息、修改密码、图书类别管理、增加图书、图书信息管理、订单管理、用户信息管理、销售信息查询和搜索等模块。

14.2　实现数据访问层

在动态网页设计中，很多都要涉及对数据库的操作，这是一件枯燥、费时而且容易出错的工作，如果使用类来实现对数据库操作的封装，就可以使写出的程序在很小的改动下完成后台数据库的更改。

14.2.1　一般数据库操作的封装

SqlAccess.cs 文件中封装了对一般数据库的操作，该文件中的主要数据库访问函数如下。

1. ExecuteNonQuery 函数

该函数执行 SQL 命令，参数为连接字符串 connString、命令类型 cmdType、存储过程

名称或 T-SQL 命令 cmdText、SQL 参数数组 cmdParms，返回值为受影响的行数 var。

```
    public static int ExecuteNonQuery(string connString, CommandType cmdType, string cmdText,
params SqlParameter[] cmdParms)
    {
        SqlCommand cmd = new SqlCommand();
        using (SqlConnection conn = new SqlConnection(connString))
        {
            PrepareCommand(cmd, conn, null, cmdType, cmdText, cmdParms);
            int val = cmd.ExecuteNonQuery();
            //清除 cmd 的参数
            cmd.Parameters.Clear();
            return val;
        }
    }
```

2. ExecuteReader 函数

该函数执行 SQL 命令，参数为链接字符串 connString、命令类型 cmdType、存储过程名称或 T-SQL 命令 cmdText、SQL 参数数组 cmdParms，返回值为 SqlDataReader 数据集。

```
    public static SqlDataReader ExecuteReader(string connectionString, CommandType commandType,
string commandText)
    {
        //pass through the call providing null for the set of SqlParameters
        return ExecuteReader(connectionString, commandType, commandText, (SqlParameter[])null);
    }
```

3. ExecuteDataset 函数

该函数通过 SqlConnection 连接执行 SQL 命令，参数为 SqlConnection 对象 connection、命令类型 cmdType、存储过程名称或 T-SQL 命令 cmdText、SQL 参数数组 cmdParms，返回值为 DataSet 数据集。

```
    public static DataSet ExecuteDataset(string connectionString, CommandType commandType, string
commandText)
    {
        return ExecuteDataset(connectionString, commandType, commandText, (SqlParameter[])null);
    }
```

4. PrepareCommand 函数

该函数完成数据库的连接和对命令的执行，供以上函数调用。

```
    private static void PrepareCommand(SqlCommand cmd, SqlConnection conn, SqlTransaction trans,
CommandType cmdType, string cmdText, SqlParameter[] cmdParms)
    {
        //判断连接的状态。如果是关闭状态，则打开
        if (conn.State != ConnectionState.Open)
            conn.Open();
```

```
        //cmd 属性赋值
    cmd.Connection = conn;
    cmd.CommandText = cmdText;
    //是否需要用到事务处理
    if (trans != null)
        cmd.Transaction = trans;
        cmd.CommandType = cmdType;
        //添加 cmd 需要的存储过程参数
        if (cmdParms != null)
        {
            foreach (SqlParameter parm in cmdParms)
                cmd.Parameters.Add(parm);
        }
    }
```

14.2.2　存储过程操作的封装

网上书店系统的商务逻辑层调用函数访问层函数，完成对 SQL Server 数据库中存储过程的操作，这是因为使用存储过程封装应用逻辑具有以下优点。

- 代码模块化，以减少数据库操作员和程序员的错误。
- 提高数据库安全性。
- 减少网络通信流量。
- 移植性较强。

本节介绍两个存储过程：AddNewBook 和 ChangeUserPWD，更多的内容请读者参照数据库。

(1) 存储过程 AddNewBook 完成向 BookInfo 数据库中添加一条图书记录的功能。输入参数为用户在页面上输入的图书名称、作者、译者、出版社、价格、内容简介、图书类别和封面。

```
USE [eBookShop]
GO
SET ANSI_NULLS OFF
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[AddNewProduct]
 (
    @productName nvarchar(50),
    @author nvarchar(50),
    @translator nvarchar(50),
    @press nvarchar(50),
    @productPrice decimal,
    @intro nvarchar(50),
    @categoryId int,
    @cover image

 )
```

```
AS
 INSERT INTO e_BookInfo
 (
      PRODUCTNAME,
      AUTHOR,
      TRANSLATOR,
      PRESS,
      PRODUCTPRICE,
      INTRO,
      CATEGORYID,
      COVER
 )
 VALUES
 (
      @productName,
      @author,
      @translator,
      @press,
      @productPrice,
      @intro,
      @categoryId,
      @cover
 )
```

(2) 存储过程 ChangePassword]完成用户密码的更改，输入参数为用户在页面上输入的旧密码、新密码和用户 ID，返回值为 1 时表示操作成功，返回值为-1 时表示操作失败。

```
USE [eBookShop]
GO
SET ANSI_NULLS OFF
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[ChangePassword]
@oldPassword nvarchar(50),
@newPassword nvarchar(50),
@userId int,
@result int output
AS
 declare @password nvarchar(50)
 select @password = userpwd from e_UserInfo where userID = @userId
 if @password = @oldpassword
 begin
      update e_UserInfo set userPwd = @newPassword where userId = @userId
      set @result = 1
 end
 else
 begin
      set @result = -1
 end
```

14.3　用户管理模块

用户管理是一个网上购物系统中不可缺少的部分。该模块主要实现用户注册、用户登录、找回丢失密码和修改个人信息等功能，主要包括以下方面。

- 用户注册页面 UserRegister.aspx，代码隐藏文件为 UserRegister.aspx.cs。
- 用户登录页面 UserLogin.aspx，代码隐藏文件为 UserLogin.aspx.cs。
- 找回丢失密码页面 UserGetPassWord.aspx，代码隐藏文件为 UserGetPassWord.aspx.cs。
- 修改个人信息页面 UserChangeInfo.aspx，代码隐藏文件为 UserChangeInfo.aspx.cs。



图 14-6　用户注册界面

14.3.1　用户注册

如图 14-6 所示的页面即为用户注册页面，运行程序后，在各项文本框中输入对应信息，然后单击【注册】按钮，即可注册新用户。

用户注册的功能有函数 Submit_Click 实现，代码如下。

```
protected void Submit_Click(object sender, System.EventArgs e)
{
  //获得注册信息
  string userName = LoginName.Text.Trim();
  string password = Password.Text.Trim();
  string question = Question.Text.Trim();
  string answer = Answer.Text.Trim();
  string userrealname=UserRealName.Text.Trim();
  string email=Email.Text.Trim();
  string role="用户";
  //得到原来的 CartId
  ShopCart cart = new ShopCart();
  string tempCartId = cart.GetShoppingCartId();
  //试图添加新用户
  User user = new User();
  int userId = user.AddNewUser(userName, password, question, answer,userrealname,email,role);
  //如果返回值为-1，则表示用户名存在
  if (userId == -1)
  {
      Message.Text = "用户名已存在！ ";
  }
  else
{
      //设置用户为通过验证
      System.Web.Security.FormsAuthentication.SetAuthCookie(userId.ToString(),false);
      //迁移购物车记录
```

```
            //将用户重定向回用户帐户页面
            Response.Redirect("UserLogin.aspx");
        }
    }
```

上述代码保存在 UserRegister.aspx.cs 文件中。先实例化 User 类，再通过调用 User 类中的 AddNewUser 函数完成用户注册，然后设置用户通过网站验证。

14.3.2　用户登录

如图 14-7 所示即为用户登录页面，在【登录名】和【密码】文本框中输入对应信息，然后单击【登录】按钮，即可完成用户 登录。

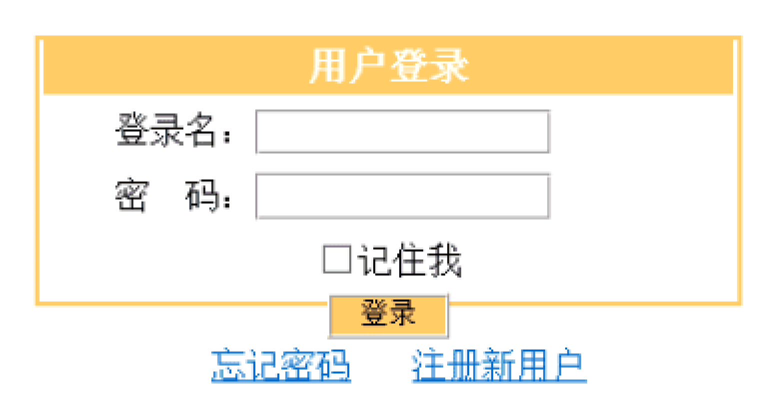


图 14-7　用户登录界面

用户登录的功能由函数 Submit_Click()实现，代码如下。

```
    protected void Submit_Click(object sender, System.EventArgs e)
    {
        User user = new eBookShop.User();
        int userId = user.Login (LoginName.Text,Password.Text);
        SqlAccess.UserRole=user.GetUserRole(userId);
        //得到临时的购物车编号
        ShopCart    cart = new ShopCart ();
        string tempCartID = cart.GetShoppingCartId();
        if (userId == 0)
        {
            Message.Text = "用户名或密码错误！";
        }
        else
        {
            //将未通过验证时该用户的匿名购物信息迁移至该用户的个人记录中
            cart.MigrateCart(tempCartID, userId.ToString());
            System.Web.Security.FormsAuthentication.RedirectFromLoginPage(userId.ToString(),
CheckCookie.Checked);
            //Response.Redirect("MyAccount.aspx");
        }
    }
```

上述代码保存在 UserLogin.aspx.cs 文件中。首先实例化 User 类，通过调用 User 类中的 Login 函数获得用户 ID。然后通过调用 User 类中的 GetUserRole 函数得到用户角色。

14.3.3　找回丢失密码

如图 14-8 所示的页面即为找回密码页面，运行程序后，在“用户名”、“提示问题”、“问题答案”和“Email”文本框中输入对应信息，然后单击“提交”按钮，可以帮助用户找回丢失密码。



图 14-8　找回丢失密码界面

用户找回丢失密码的功能由函数 Submit_Click()实现，代码如下。

```
protected void Submit_Click(object sender, System.EventArgs e)
{
  User user = new eBookShop.User();
  int result   = user.GetBackPassword(LoginName.Text.Trim(), Question.Text.Trim(),
      Answer.Text.Trim(), Email.Text);
  if (result == 1)
  {
      Message.Text = "您的密码已发送，请到邮箱查收";
      //user.ChangePassword(
  }
  else
  {
      Message.Text = "您的输入信息有误！";
  }
}
```

上述代码保存在 UserGetPassWord.aspx.cs 文件中。首先实例化 User 类，然后通过调用 User 类中的 GetBackPassword 函数为用户发送新的 6 位随机密码。

User 类中的 GetBackPassword 函数的具体内容如下。

```
public int GetBackPassword(string userName, string question, string answer, string email)
{
        object m_DBNull = Convert.DBNull;
        //获得新的随机密码
        string newPassword = MakePassword(6);
        //定义存储过程参数
        SqlParameter[] para = {
                                new SqlParameter("@userName", userName),
                                new SqlParameter("@question", question),
                                new SqlParameter("@answer", answer),
                                new SqlParameter("@newPassword", newPassword),
                                new SqlParameter("@result", SqlDbType.Int, 8,
ParameterDirection.Output,
                                true, 0, 0, "", DataRowVersion.Default, m_DBNull)
                                            };
             //执行存储过程
             try
             {
             SqlAccess.ExecuteNonQuery(SqlAccess.CONN_STRING,
CommandType.StoredProcedure,"GetBackPwd", para);
```

```
                }
                catch
                {
                    throw new Exception("邮件无法发送！");
                }
                //获得输出参数的值
                int result = Convert.ToInt32(para[4].Value);
                //如果密码保护资料填写正确
                if (result == 1)
                {
                    //从 Web.config 获取发信人地址、邮件标题、邮件用户名和密码以及 SmtpServer
    //              string sender =
System.Configuration.ConfigurationSettings.AppSettings["mainSender"];
                    try
                    {
                        MailMessage mail = new MailMessage();
                        mail.To = email;
                        mail.From = "bao@163.com" ;
                        mail.Subject = "您在 eBookShop 的密码已找回";
                        mail.BodyFormat = MailFormat.Text ;
                        mail.Body = "您在 eBookShop 的密码已找回,新密码为"+newPassword  ;
                        SmtpMail.Send(mail) ;
                    }
                    catch(Exception ex)
                    {
                        throw new Exception(ex.Message);
                    }
                }
                return result;
            }
```

上述代码保存在 User.cs 文件中，通过调用 MakePassword 产生 6 位新的随机密码。调用存储过程 FindPassword 将新密码写入数据库 User 表中。最后定义一个 MailMessage 实例 mail，填写所需的目的地址、发送人地址、主题、内容格式、内容，通过 SmtpMail.Send(mail) 进行发送。

14.3.4　修改个人信息

如图 14-9 所示即为修改个人信息页面，运行程序，在【真实姓名】、【问题】、【答案】、Email 文本框中输入对应信息，然后单击【提交】按钮，可以完成用户个人信息的修改。



14-9　修改个人信息界面

修改个人信息的功能由函数 UpdateUserInfo()实现，代码如下。

```
    void UpdateUserInfo()
    {
        User user = new eBookShop.User();
        user.ChangeProfile(User.Identity.Name, RealName.Text,
```

```
            Question.Text,Answer.Text , Email.Text);
    }
```

上述代码保存在 UserInfo.aspx.cs 文件中。

本程序中实例化 User 类，通过调用 User 类中的 ChangeProfile 函数更新用户信息。ChangeProfile 函数调用存储过程 EditUserInfo，其具体内容如下。

```
    public int ChangeProfile(string userId, string userRealName, string question, string answer, string
email)
            {
                SqlParameter[] para = {new SqlParameter("@userId", int.Parse(userId)),
                                        new SqlParameter("@userRealName", userRealName),
                                        new SqlParameter("@question", question),
                                        new SqlParameter("@answer", answer),
                                        new SqlParameter("@email", email)
                                        };
                return SqlAccess.ExecuteNonQuery(SqlAccess.CONN_STRING,
CommandType.StoredProcedure, "EditUserInfo", para);
    }
```

14.4　图书管理模块

图书管理是系统的重要组成部分，它主要实现管理图书类别，添加、删除和修改图书信息，列表显示图书，搜索图书等功能。下面将介绍这 4 部分功能的实现方法。

14.4.1　管理图书类别

图书类别管理页面 ManBookType.aspx 的设计如图 14-10 所示。运行程序后，在【图书类型名称】文本框中输入图书类型信息，然后单击【添加】按钮，可以完成图书类型的添加。

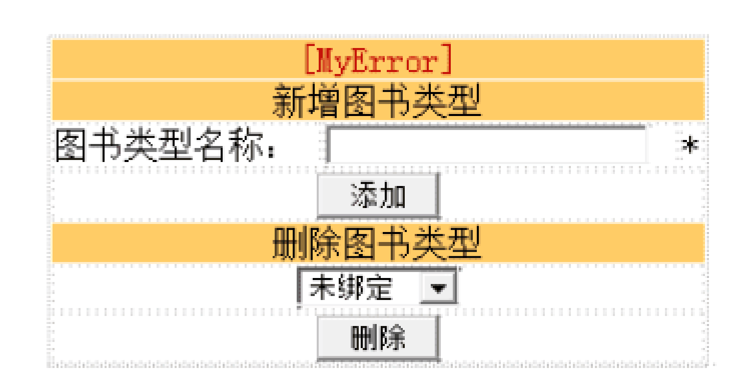


图 14-10　管理图书类别

添加图书类型的功能由函数 Add_Click()实现，代码如下。

```
    protected void Add_Click(object sender, System.EventArgs e)
    {
      AdminDB admin = new AdminDB();
      try
      {
          admin.AddNewCategory(CategoryName.Text.Trim());
          MyError.Text = "添加成功！ ";
          BindList();
      }
      catch
      {
```

```
            MyError.Text = "出错了！";
        }
    }
```

上述代码保存在 ManBookType.aspx.cs 文件中。首先实例化 Admin 类，再通过调用 Admin 类中的 AddNewCategory 函数添加新的图书类别。

在图 14-10 所示的【删除图书类型】下拉列表框中选择图书类型，然后单击【删除]按钮，可以完成图书类型的删除。删除图书类型的功能由函数 Delete_Click()实现，代码如下。

```
    protected void Delete_Click(object sender, System.EventArgs e)
    {
     AdminDB admin = new AdminDB();
     try
     {
            admin.DeleteCategory(int.Parse(ListCat.SelectedValue));
            MyError.Text = "删除成功！";
            BindList();
     }
     catch
     {
            MyError.Text = "出错了！";
     }
    }
```

上述通过调用 Admin 类中的 DeleteCategory 函数删除图书类别。



图 14-11　添加图书

14.4.2　添加、删除、修改图书信息

如图 14-11 所示即为新增图书页面 ManBookInfo.aspx 的设计效果。运行程序后，输入相应信息，并单击【浏览】按钮选择图书封面后确定，然后选择图书类别，单击【添加】按钮，即可完成新图书的添加。

图书添加的功能由函数 Add_Click()实现，代码如下。

```
    protected void Add_Click(object sender, System.EventArgs e)
         {
              int ImageSize=0;
              byte[] ImageBuffer;
              string FileName;
              FileInfo TpFile;
              FileStream fs=null;
    FileName=Server.MapPath("\\eBookShop\\CoverUp").ToString().Trim()+"\\"+ViewState["PostFileNa
me"];
              TpFile=new FileInfo(FileName);
```

```
                ImageSize=(int)TpFile.Length;
                fs=new FileStream(FileName,FileMode.OpenOrCreate);
                BinaryReader r=new BinaryReader(fs);
                r.BaseStream.Seek(0,SeekOrigin.Begin);
                ImageBuffer=r.ReadBytes(ImageSize);
                AdminDB admin = new AdminDB();
                try
                {
    admin.AddNewProduct(ProductName.Text.Trim(),Author.Text.Trim(),Translator.Text.Trim(),Press.T
ext.Trim(), decimal.Parse(Price.Text.Trim()), Intro.Text, int.Parse(ListCat.SelectedValue),ImageBuffer);
                    MyError.Text = "添加成功！ ";
                }
                catch
                {
                    MyError.Text = "出错了";
                }
            }
```

上述代码保存在 ManBookInfo.aspx.cs 文件中。从图片路径中将图片以二进制文件流的形式赋值给 ImageBuffer，然后实例化 Admin 类，通过调用 Admin 类中的 AddNewProduct 函数添加图书信息。

在如图 14-12 所示的【图书名称】、【作者】、【译者】、【出版社】、【价格】、【介绍】文本框中修改相关信息，然后单击【修改】按钮，可以完成图书信息的修改。

图书列表

编号	图书名称	作者	译者	出版社	价格	介绍	删除
17	MIPS处理器设计透视 Se	Dominic Sweetman	赵俊良	北京航空航天大学出版社	55	本书涵盖了有关MIPS处理器的发展历史、运行原理和	☐
18	PIC单片机基础教程	丁跃军		北京航空航天大学出版社	17	本书以Microchip公司的PIC16F87X单片机为主	☐
19	电子线路与电子技术	张洪润		清华大学出版社	32	本书根据电子技术基础课程教学大纲的要求而编写。全	☐
20	计算机系统结构教程	尹朝庆		清华大学出版社	26	本书以提高计算机并行性的方法和技术为主线，以性能	☐
21	CMOS大规模集成电路设	尼尔 H.E.威斯特		机械工业出版社	125	本书详细介绍了在一个芯片上设计复杂和高性能CMOS	☐
22	多机器人系统	谭民		清华大学出版社	39	在本书的编写过程中得到了许多人的支持和帮助，在	☐
23	机器人学导论（英文影印	约翰 J.克拉格		机械工业出版社	49	本书源于斯坦福大学1983年至1985年秋季学期的"机	☐
24	智能控制	李少远		机械工业出版社	20	本书从控制系统建模、控制与优化的本质要求出发，系	☐
25	C#大学教程		须德 等/译	电子工业出版社	59	本书是Deitel公司的“Deitel开发者丛书”	☐
26	C#行家设计手册	王浩然		中国铁道出版社	50	C#（C Sharp）是一种最新的面向对象的编程语	☐

1 2

修改

图 14-12　编辑修改图书

如果选中【删除】复选框，则单击【修改】按钮后可以完成该条记录的删除。程序员图书修改、删除的功能由函数 UpdateProductInfo()实现，代码如下。

```
    void UpdateProductInfo()
        {
            AdminDB admin = new AdminDB();
            for (int i=0; i< GridProduct.Rows.Count; i++)
```

```
            {
                int productId = Convert.ToInt32(GridProduct.DataKeys[i]);
                string productName = ((TextBox)
GridProduct.Rows[i].FindControl("ProductName")).Text.Trim();
                decimal price = decimal.Parse(((TextBox)
GridProduct.Rows[i].FindControl("Price")).Text.Trim());
                string intro = ((TextBox) GridProduct.Rows[i].FindControl("Intro")).Text;
                string author = ((TextBox) GridProduct.Rows[i].FindControl("Author")).Text;
                string translator = ((TextBox)
GridProduct.Rows[i].FindControl("Translator")).Text;
                string press = ((TextBox) GridProduct.Rows[i].FindControl("Press")).Text;
                CheckBox remove = (CheckBox) GridProduct.Rows[i].FindControl("Remove");

                try
                {
                    if (remove.Checked)
                    {
                        admin.DeleteProduct(productId);
                    }
                    else
                    {
                        admin.AlterProduct(productId, productName, price,
intro,author,translator,press);
                    }
                    Message.Text = "操作成功！ ";//提示成功
                }
                catch
                {
                    Message.Text = "输入有误！ ";
                }
            }
        }
```

上述代码保存在 ManBookList.aspx.cs 文件中。首先读取 GridView 中管理员修改的信息，如果选中【删除】复选框，则调用 Admin 类中的 DeleteProduct 方法删除记录；如果没有选中【删除】复选框，则调用 Admin 类中的 AlterProduct 方法更新图书信息。

14.4.3　列表显示图书

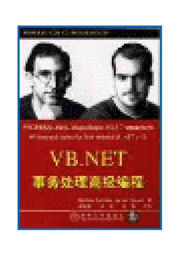
如图 14-13 所示为按类别列表显示图书信息的界面。



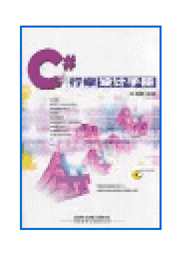
VB.NET大学教程-国际版
作者:
出版社:清华大学出版社
价格:$65
加入购物车



XML与JAVA程序设计大全
作者:台湾金禾资讯
出版社:中国铁道出版社
价格:$46
加入购物车



VB.NET事务处理高级编程
作者:Matthew Bortniker James Conard
出版社:清华大学出版社
价格:$52
加入购物车



C#行家设计手册
作者:王浩然
出版社:中国铁道出版社
价格:$50
加入购物车

图 14-13　列表显示图书

图书列表显示的功能由函数 DataListBind()实现，代码如下。

```
    private void DataListBind()
    {
     dl_BookByType.DataSource=Product.GetBookByCategory(int.Parse(ViewState["BookTypeId"].ToS
tring()) );
    dl_BookByType.DataBind();
    }
```

上述代码保存在BookByType.aspx.cs文件中。程序调用Book类中的GetBookByCategory函数实现图书信息获取，该函数具体内容如下。

```
    public static SqlDataReader GetBookByCategory(int BookTypeId )
    {
            string strSql;
            strSql="select ProductId,productName,Author,Translator,Press,ProductPrice,CategoryId,Intro=
'  '+SUBSTRING(Intro,0,500)+'.....',image=case when(not Cover is null) then ' <img src=
BookCover.aspx?ProductId='+cast(ProductId as varchar(10))+' Border=1 width=70 height=100>' else '
<img src=img/pic.jpg border=1 width=70 height=100>' end from e_BookInfo where CategoryId="
+BookTypeId;
            return SqlAccess.ExecuteReader(SqlAccess.CONN_STRING,
            CommandType.Text, strSql);
            }
```

程序中 strSql 字符串的函数是：从 BookInfo 表中选择各项信息，满足类型号等于传出的类型号，其中的 Image 若不为空，则调用 BookCover.aspx 页面完成二进制字符到图像的转换，如果为空则指向默认的图片。

14.4.4　搜索图书

搜索图书的执行流程如图 14-14 所示。如图 14-15 所示为图书搜索页面设计视图，运行程序后，在【图书名称】、【作者】、【译者】、【出版社】文本框中输入对应信息，选择图书种类，然后单击【查找】按钮以实现搜索功能。

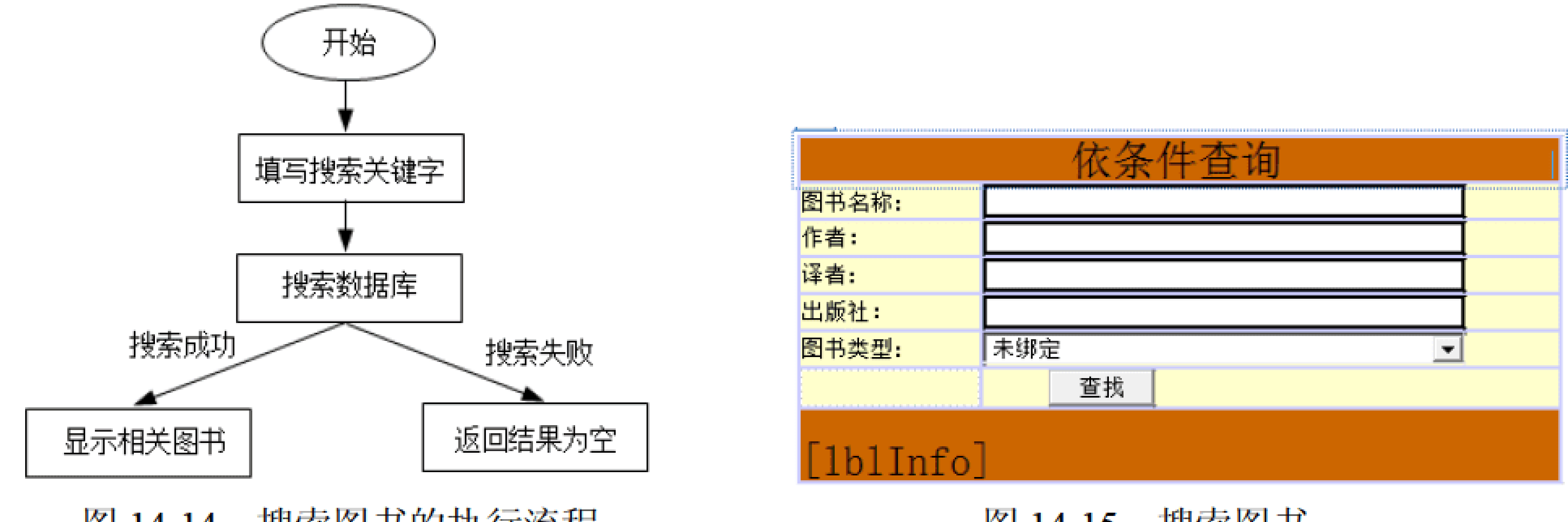


图 14-14　搜索图书的执行流程　　图 14-15　搜索图书

程序首先调用函数 Search_Click()实现，然后调用 BookSearchResult 页面中的 Bind()函数，通过 Book 类的 Search 方法执行数据库操作命令。函数 SearchBtn_Click ()代码如下。

```
    protected void SearchBtn_Click(object sender, System.EventArgs e)
```

```
{
            string Name = this.txtName.Text;
            string Author = this.txtAuthor.Text.Trim();
            string Translator = this.txtTranslator.Text.Trim();
            string Publisher = this.txtPublisher.Text.Trim();
            string TypeId = this.ListCat.Items[this.ListCat.SelectedIndex].Value;
            string str = "";
            if(int.Parse(TypeId)>=0)
            {
                str += " And CategoryId=" + TypeId;
            }
            if(Author.Length>0)
            {
                str += " And Author Like '%" + Author + "%'";
            }
            if(Name.Length>0)
            {
                str += " And ProductName Like '%" + Name + "%'";
            }
            if(Translator.Length>0)
            {
                str += " And Translator Like '%" + Translator + "%'";
            }
            if(Publisher.Length>0)
            {
                str += " And Press Like '%" + Publisher + "%'";
            }
            Response.Redirect("BookSearchResult.aspx?Condition=" + str);
        }
}
```

上述代码保存在 BookSearch.aspx.cs 文件中。程序读取用户在各文本框中输入的查询信息，组合成查询字符串，然后重定向到 BookSearchResult.aspx 文件，并将查询条件以参数 Condition 的形式传递给它。图 14-16 所示以列表的形式显示搜索结果。图书具体搜索的功能又函数 Bind()实现。

图书名称	图书类型	作者	译者	出版社	价格
MIPS处理器设计透视 See MIPS Run	13	Dominic Sweetman	赵俊良	北京航空航天大学出版社出版	55
PIC单片机基础教程	13	丁跃军		北京航空航天大学出版社出版	17

图 14-16　搜索结果

函数 Bind()代码如下。

```
private void Bind()
{
        DataSet ds = Product.Search(Request["Condition"]);
        //lblInfo.Text = "Total results:" + ds.Tables[0].Rows.Count.ToString();
        this.GridView1.DataSource = ds.Tables[0].DefaultView;
        this.GridView1.DataBind();
}
```

上述代码保存在 BookSearchResult.aspx.cs 文件中，其中调用的 Book 类中的 Search 方法内容如下。

```
public static DataSet Search(string strCondition)
{
  string strSQL =   "Select * From e_BookInfo Where 1=1 " + strCondition + " Order by ProductId";
  return SqlAccess .ExecuteDataset(SqlAccess.CONN_STRING, CommandType.Text,strSQL);
}
```

程序将查询结果以列表形式显示。

14.5　购物车模块

购物车模块使用户可以在购物时，随时查看自己购买了多少商品，且可以随时添加和删除所购买的商品。

14.5.1　查看购物列表

查看购物列表的执行流程如图 14-17 所示。用户可以通过单击图书旁边的【加入购物车】按钮，将图书加入购物车。图 14-18 所示显示了用户购物车，具体包括图书编号、图书名称、价格、数量、小计和总计等内容。

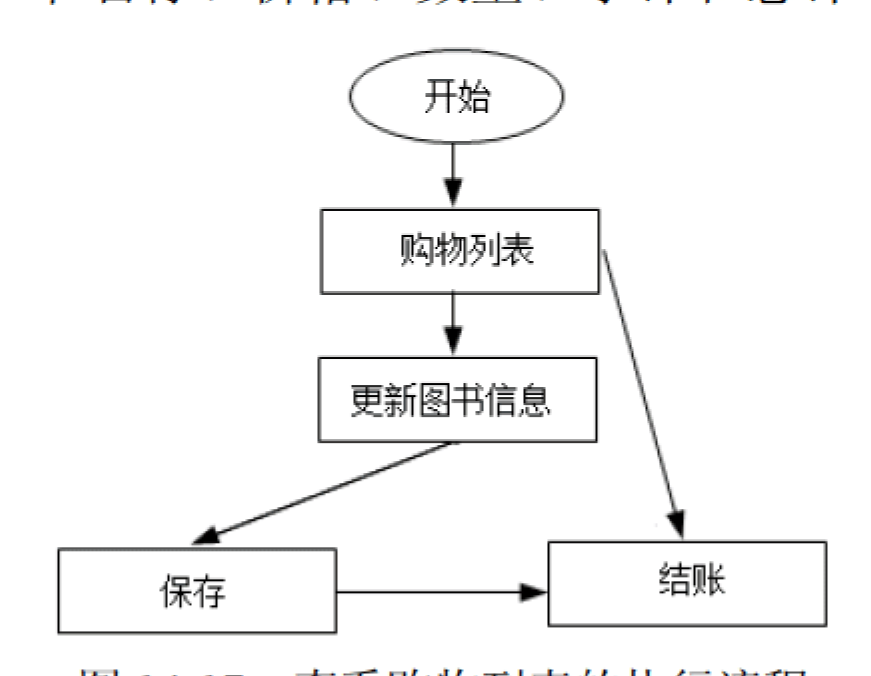


图 14-17　查看购物列表的执行流程



图 14-18　查看购物列表

购物列表显示功能由函数 ShowShopCart()实现，代码如下。

```
void ShowShopCart()
{
        ShopCart cart = new ShopCart();
        // 得到用户的购物车 ID
        String cartID = cart.GetShoppingCartId();
        // 如果购物车内没有商品,GridView 隐藏
        if (cart.GetItemCount(cartID) == 0)
        {
            DetailsPanel.Visible = false;
            MyError.Text = "购物车内没有商品。";
        }
        else
```

```
                {
                        // 绑定购物车信息到 GridView
                        MyList.DataSource = cart.GetItems(cartID);
                        MyList.DataBind();
                        //显示总金额
                        lblTotal.Text = String.Format( "{0:c}", cart.GetTotal(cartID));
                }
        }
```

上述代码保存在MyShopCart.aspx.cs文件中。实例化ShopCart对象cart，调用ShopCart类中的GetShopCartId方法，得到cartID。调用GetShopCartCount方法得到总金额，再调用GetShopCartItem方法得到购物车条目。

14.5.2　结账

用户结账的执行流程如图 14-19 所示。图 14-20 所示为购物车结账界面，结账功能的实现与前面查看购物列表功能的实现相似，请读者参考上述内容。



图 14-19　用户结账的执行流程　　　图 14-20　为购物车结账界面

14.6　订单管理模块

凡是网上购物，就必须下订单。订单是观察所购买商品的一个预览表格，这里可以显示你所买的商品、价格和日期等。

14.6.1　客户订单管理

查询用户信息的执行流程如图 14-21 所示。如图 14-22 所示为客户所提交的订单，包括订单号、订单时间、订单总额等信息。

客户订单浏览功能在函数 Page_Load()中实现，代码如下。

```
        protected void Page_Load(object sender, System.EventArgs e)
        {
                if (!Page.IsPostBack)
                {
                 // ShowMenu();
```

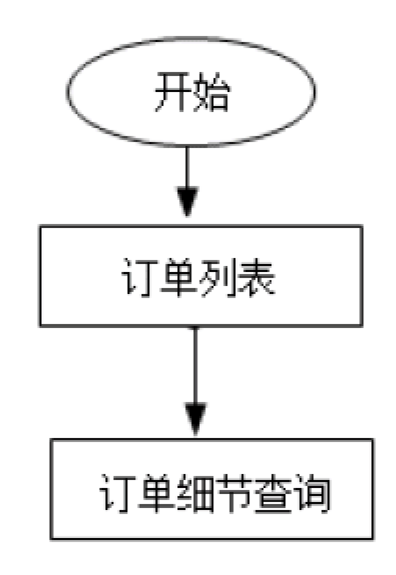


图 14-21　查询用户信息的执行流程

订单号	订单时间	订单总额	查询详细信息
7	2015/2/10	¥111.00	查询
8	2015/2/10	¥111.00	查询
9	2015/2/10	¥188.00	查询
10	2015/2/10	¥50.00	查询

图 14-22　客户订单浏览

```
        }
        string userID = User.Identity.Name;
        Orders orderHistory = new Orders();
        MyList.DataSource = orderHistory.GetOrderList(userID);
        MyList.DataBind();
        if (MyList.Rows.Count == 0)
        {
            MyList.Visible = false;
            MyError.Text = "您没有订单记录";
        }
    }
```

上述代码保存在 MyOrder.aspx.cs 文件中。首先读取当前用户 ID 赋值给 userID，再实例化为 Orders 类，通过调用 Orders 类中的 GetOrderList 函数得到用户订单信息并传给列表显示。单击【查询】链接可以查看具体某条订单细节，界面如图 14-23 所示。

您的订单号是: 7
订单日期为: 2015/2/10

图书编号	图书名称	价格	数量	小计
27	XML与JAVA程序设计大全	¥46.00	1	¥46.00
31	VB.NET大学教程-国际版	¥65.00	1	¥65.00

总计: ¥111.00

图 14-23　客户订单细节浏览

14.6.2　管理员订单管理

订单管理的执行流程如图 14-24 所示。图 14-25 所示为管理员显示了全部客户已经提交的订单，该页面提供了具体订单浏览及订单处理的功能。

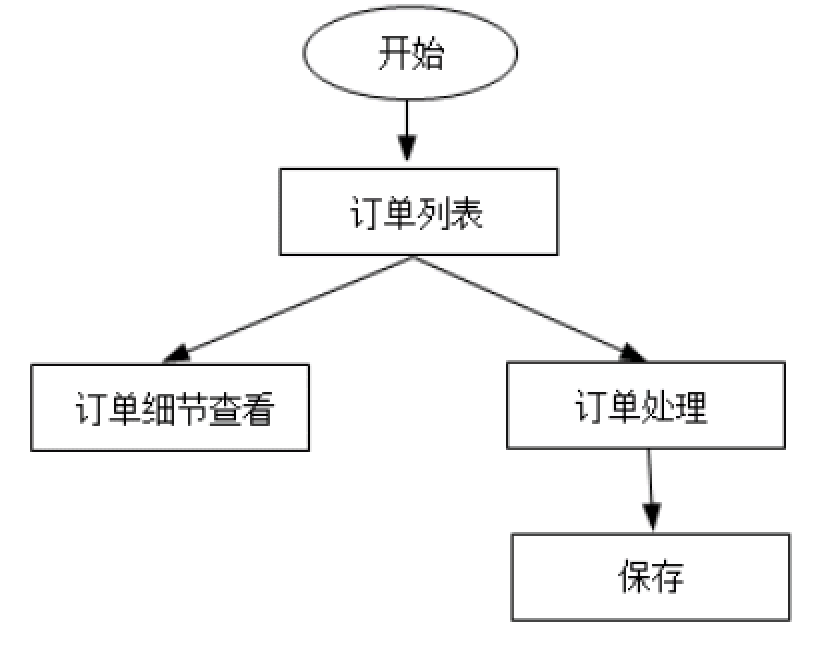


图 14-24　订单管理的执行流程

管理订单
您共有 4 条订单。

订单号码	订单时间	用户号码	状态	处理
7	2015/2/10 15:06:20	2	已处理	✔
8	2015/2/10 15:06:36	2	已处理	✔
9	2015/2/10 15:07:17	2	已处理	✔
10	2015/2/10 15:07:33	2	未处理	✔

图 14-25　管理员订单浏览

客户订单显示的功能在函数 Bind()中实现，代码如下。

```
private void Bind()
{
        try
        {
            Orders    orders = new Orders ();
            DataSet ds =orders.GetAllOrder();
            //ds.Tables[0].DefaultView.Sort = ViewState["Sort"].ToString();
            this.GridView1.DataSource = ds.Tables[0].DefaultView;
            this.GridView1.DataBind();
            this.Message.Text = "您共有  " + ds.Tables[0].Rows.Count.ToString() + "  条订单。";
        }
        catch
        {
            this.Message.Text="发生错误。";
        }
}
```

上述代码保存在 ManOrder.aspx.cs 文件中，首先实例化 Orders 对象，然后通过协调 Orders 类的 GetAllOrder 方法获得全部订单信息，并在列表中显示。

该页面还为管理者提供了订单处理的功能，在函数 GridView1_EditCommand()中实现，代码如下。

```
protected void GridView1_EditCommand(object source, GridViewEditEventArgs e)
{
        try
        {
            string oid = this.GridView1.DataKeys[e.NewEditIndex].ToString();
            Orders    orders = new Orders ();
            orders.DealOrder(oid);
            Bind();
        }
        catch
        {
                Message.Text = "发生错误"; //提示失败
        }
}
```

上述代码首先实例化 Orders 对象，再调用 Orders 类的 DealOrder 方法对所选择订单进行处理，列表中相关信息显示为“已处理”。

14.7　后台管理模块

后台管理模块包含查询用户信息和查询销售信息，可以通过查询用户信息来增加管理员，通过查询销售信息从总体上了解图书销售数量及金额。

14.7.1　查询用户信息

查询用户信息的执行流程如图 14-26 所示。如图 14-27 所示为管理员提供了用户列表显示及角色管理的功能。



	用户ID	用户登录名	真实姓名	角色	Email
	1	Landy	兰迪	管理员	Landy@163.com
	2	bao2	兰迪2	用户	Landy@163.com
	5	bao3	兰迪3	用户	Landy@123.com

图 14-26　查询用户信息的执行流程　　　　图 14-27　用户管理

角色管理功能由函数 GridUsers_UpdateCommand()实现，代码如下。

```
protected void GridUsers_UpdateCommand(object source, GridViewUpdateEventArgs e)
        {
            string role =
((TextBox)GridUsers.Rows[e.RowIndex].FindControl("Role")).Text.Trim();

           //获得要更改的用户的 UserId
            int userId = Convert.ToInt32(GridUsers.DataKeys[e.RowIndex]);
           AdminDB admin = new AdminDB();
           admin.UpdateUserRole(role, userId.ToString());
           //退出编辑状态
           GridUsers.EditIndex = -1;
           //绑定
           BindGrid();
        }
```

上述代码保存在 ManUser.aspx.cs 文件中，首先实例化 Admin 对象，再调用 Admin 类中的 UpdateUserRole 方法完成用户角色更改。

14.7.2　查询销售情况

查询销售情况的执行流程如图 14-28 所示。图 14-29 所示的查询销售情况页面为管理员提供了按月、日查询图书销售种类、数量、金额的功能。

查询销售情况的功能主要由函数 BindGrid()实现，代码如下。

```
void BindGrid(string year, string month, string day)
{
 GridSails.DataSource = new AdminDB().GetSails(year, month, day);
 GridSails.DataBind();
}
```

上述代码保存在 ManSales.aspx.cs 文件中，其中 Admin 类中的 GetSails 方法内容如下。

```
public  DataSet GetSails(string year, string month, string day)
```

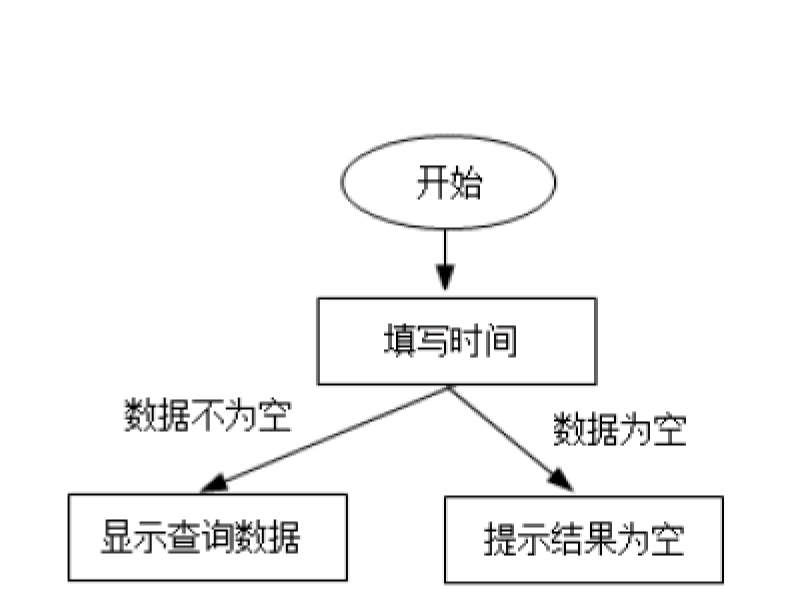


图 14-28　查询销售情况的执行流程

查询销售信息

2015 年 2 月 10 日

查询月销售情况　查询日销售情况

图书编号	图书名称	相关订单数	售出数量	销售收入
28	21天学通J2EE（1CD）	1	2	168
31	VB.NET大学教程-国际版	1	1	65
29	Java与CORBA客户/服务器编程(第二版)1CD	1	1	59
30	VB.NET事务处理高级编程	1	1	52
26	C#行家设计手册	1	1	50
27	XML与JAVA程序设计大全	1	1	46
24	智能控制	1	1	20

图 14-29　查询销售情况

```
    {
    SqlParameter[] para = {new SqlParameter("@year", int.Parse(year)),new SqlParameter("@month",
int.Parse(month)),new SqlParameter("@day", int.Parse(day)) };
    return SqlAccess.ExecuteDataset(SqlAccess.CONN_STRING, CommandType.StoredProcedure,
"GetSails", para);
    }
```

上述代码首先接受传入的参数月、日、年，然后执行 GetSails 存储过程得到查询的销售情况。

14.8　技术亮点

本章为读者介绍了电子商务网站开发过程中表示层、商务逻辑层和数据访问层的设计结构。

网上书店的难点在于程序的流程，本章通过介绍网站的各项功能，清晰简洁地展现了网上书店的开发流程，这一流程可以应用于各类网上购物系统，如网上花店、网上商店等。

在图书信息添加、显示模块中，以二进制格式上传、下载图片文件，是 ASP.NET 的一个技术难点，也是网站开发中的一个重点。这其中应用了文件对象操作，相应部分的代码可以在其他程序中重复应用。

GridView 控件、DataList 控件和 Repeater 控件的使用，使数据显示和更新更加多样化、灵活化、丰富化。

14.9　本章小结

本章介绍了目前互联网中比较经典的电子商务网站——网上书店系统，其中实现了用户注册、登录、个人信息修改、图书信息管理、购物车管理、订单管理、销售查询等功能。这些功能是网上购物系统中不可缺少的部分。本章较详细地介绍了实现各项功能的商务逻辑层和数据访问层的代码，并做了相应注释和讲解。表现层和数据层的详细内容请读者参

考光盘中的源程序。

14.10　思考与练习

1. 互联网中国内比较知名的网上书店有哪些？
2. 对 1 题中的网上书店的分类进行调研。

参 考 文 献

[1] 明日科技. ASP.NET 从入门到精通[M]. 3 版. 北京：清华大学出版社，2013.

[2] 蒋金楠. ASP.NET MVC 5 框架揭秘[M]. 北京：电子工业出版社，2014.

[3] [美] 盖洛德(Jason N.Gaylord)，[美] Pranav Rastogi，[美] Todd Miranda，等 著； 李增民，苗荣 译. ASP.NET 4.5 高级编程[M]. 8 版. 北京：清华大学出版社，2014.

[4] [美] Imar Spaanjaars 著；刘楠，陈晓宇 译. ASP.NET 4.5 入门经典[M]. 7 版. 北京：清华大学出版社，2013.

[5] 高宏等. ASP.NET 典型模块与项目实战大全[M]. 北京：清华大学出版社，2012

[6] 张正礼. ASP.NET 4.0 网站开发与项目实战(全程实录). 北京：清华大学出版社，2012.

[7] 国家 863 中部软件孵化器. 编程宝典：ASP.NET 从入门到精通[M]. 北京：人民邮电出版社，2010.

[8] Scott Millett 著；杨明军 译. ASP.NET 设计模式[M]. 北京：清华大学出版社，2011.

[9] 朱元波. ASP.NET 项目开发详解[M]. 北京：清华大学出版社，2014.

[10] 张正礼，王坚宁. ASP.NET 4.0 从入门到精通[M]. 北京：清华大学出版社，2011

[11] 赛奎春，顾彦玲. ASP.NET 项目开发全程实录[M] . 3 版. 北京：清华大学出版社，2013.

[12] 张昌龙，辛永平. ASP.NET 4.0 从入门到精通[M]. 北京：机械工业出版社，2011.

[13] 斯蒂芬森(Ryan Stephens)，等 著； 井中月，郝记生 译. SQL 入门经典[M]. 5 版. 北京：人民邮电出版社，2011.

[14] [日] MICK 著；孙淼，罗勇 译. SQL 基础教程[M]. 北京：人民邮电出版社，2013.

[15] [英] Charles Wyke-Smith 著；李松峰 译. CSS 设计指南[M]. 3 版. 北京：人民邮电出版社，2013.

[16] 陶国荣. jQuery 权威指南[M]. 2 版. 北京：机械工业出版社，2013.

[17] 张欣毅. XML 简明教程[M]. 北京：清华大学出版社，2009.

[18] [美] 达克特(Jon Duckett) 著；王德才，吴明飞，姜少猛 译. HTML、XHTML、CSS 与 JavaScript 入门经典[M]. 北京：清华大学出版社，2011.